Tiago Miguel Rebouço Pina

Licenciado em Engenharia Electrotécnica e de Computadores

Development of Behavioral Models for Memristors

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Orientador: Prof. Doutora Maria Helena Fino, Professora Auxiliar, FCT-UNL

Júri:

Presidente: Prof. Doutor João Rosas, Professor Auxiliar, FCT-UNL

Arguente: Prof. Doutor Pedro Pereira, Professor Auxiliar, FCT-UNL

Vogal: Prof. Doutora Maria Helena Fino, Professora Auxiliar, FCT-UNL

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro 2018

Development of Behavioral Models for Memristors

Aos meus pais e à minha avó
que nunca me abandonaram

Acknowledgments

I would like to address my gratitude to my advisor, Professora Maria Helena Fino for all the patience and guidance over this year. The motivation and knowledge transmitted were the key to this work being completed. I would like to thank the opportunity that Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa gave to me, to finish my academic course.

In addition, to my friends who accompanied me until here and those I met in this journey, I express all my gratitude. For all my family that has always been with me, specially my parents, thank you so much. A special mention for all those who have passed away over the years, namely my grandparents, who taught me so many things giving to me the small steps to become the person I am today. They will never be forgotten.

Without all this support it would be impossible to complete this important goal. Thank you so much!

Abstract

The memristors are electronic devices that have been introduced as passive elements, where the magnetic flux between their two terminals is a function of the total electrical charge that passes through the element. These elements can be used in various applications such as logic circuits or non-volatile memories.

In this work we intend to develop behavioral models memristors that, after being tested and validated, allow the implementation of oscillator circuits. A first implementation of macromodels in LTspice was considered in order to verify the non-linear behaviors presented by this device.

Due to this type of behavior, particularly at the borders of this element, the introduction of window functions is presented by modeling the best way its operation. The presented models were developed in Matlab environment, using the Euler method to solve differential equations. The implementation of this method presents numerical approximation errors, causing the hysteresis loop for several periods to not coincide. The introduction of Runge-Kutta method of order 4.5 is presented, where by using a variable step it is possible to present better results at the computation level.

The validated models were introduced in VerilogA environment in order to demonstrate a small example of an oscillator based on memristors without using reactive elements.

Keywords: Memristor, window functions, Euler method, Runge-Kutta method

Resumo

Os memristores são dispositivos electrónicos que foram introduzidos como sendo elementos passivos, em que o fluxo magnético entre os seus dois terminais é função do total de carga eléctrica que passa pelo elemento. Estes elementos podem ser utilizados em várias aplicações como, por exemplo, circuitos lógicos ou memórias não voláteis.

Neste trabalho pretendeu-se desenvolver modelos comportamentais de memristores que depois de testados e validados permitam a implementação em circuitos osciladores. Uma primeira implementação de macromodelos em LTspice foi considerada, de forma a verificar os comportamentos não lineares apresentados por este dispositivo.

Devido a este tipo de comportamentos, nomeadamente nas fronteiras deste elemento, a introdução de funções janela é apresentada modelando da melhor forma o seu funcionamento. Os modelos apresentados foram desenvolvidos em ambiente Matlab, utilizando o método de Euler para resolução de equações diferenciais. A implementação deste método apresenta erros de aproximação numérica, fazendo com que os loops de histerese para vários períodos não coincidam. A introdução do método de Runge-Kutta de ordem 4.5 é apresentada, onde utilizando um passo variável é possível apresentar resultados melhores ao nível de computação.

Os modelos validados foram introduzidos em ambiente VerilogA de forma a demonstrar um pequeno exemplo de um oscilador baseado em memristores sem recurso a elementos reactivos.

List of Contents

List of Tables

List of Figures

1. Introduction

1.1 Motivation

Over the years, we have seen a significant evolution in the electronics industry. This industry has evolved in such a way, that today it is impossible to live without access to electronic devices.

The passive elements; Resistor, Capacitor and Inductor have long been considered the fundamental elements. In 1971 Leon Chua discovered a fourth relevant element, which he called a memristor [1]. Memristor is defined as “memory resistor”, which subtend that information could be storage [2]. The information storage is based on the value of resistance that memristor holds. This resistance changes under toggle conditions, e.g., an applied current or voltage [3]. The pinched hysteresis loop, which is an intrinsic feature, depends on the frequency of the applied input signal. This type of memory devices is used in many applications such as oscillators, programmable analog circuits and non-volatile memories [1].

1.2 Objectives

The main objective in this work is to develop behavioral models for memristors. The first step is to study the operation of these devices and their application in different areas, of analog design. A first implementation in LTspice is considered. The implemented models will be validated using the software Matlab script language.

To be integrated in Cadence environment, the models will be converted to Verilog-A language and then used in more complex circuits, such as oscillators, allowing their simulation.

This dissertation led to the submission of the following document:

M. H. Fino and T. Pina, “On the use of modified Biolek window for Memristor modeling in VerilogA,” 2018.

1.3 Organization

The dissertation is organized as follows: section 2 presents the state of the art, which describes the two types of models, e.g. linear and nonlinear. The linear ion drift models need to be developed with some window functions to accurately mimic to memristor behavior. The nonlinear ion drift models show a behavior closer to reality, that is, the memristors behave as nonlinear elements.

In section 3, a first implementation of macromodels in LTspice is presented. These macromodels were based on the linear models and tested with the several window functions. For a validation of these models, an implementation using the MatLab software script language was made. The introducing of the Runge-Kutta method to the implemented memristor models is presented to compare with the Euler method implemented in the previous sections.

Section 4 shows a possible implementation of a memristor-based reactance-less oscillator. The idea is to replace the reactive elements with the implemented memristor models.

Finally, in section 5, the conclusions and future remarks of this work are presented.

2. Memristors

This chapter begins with an introduction defining a memristor, their constitution and how they work. Section 2.2 present the existing linear ion drift models for this device and how can be defined. In section 2.3, a several window functions that will be implemented in this work are presented. Finally, the nonlinear ion drift models are introduced in section 2.4, which are the best models to accurate by mimic the memristors behavior.

2.1 Introduction

Memristors are novel electronic devices introduced in 1971 by professor Chua. Later these new devices were implemented by HP Laboratories in 2008 and a linear ion drift model was proposed [4]. These devices are basically resistors with memory and are included in group of passive devices [2]. The main difference between these devices and resistors are the nonlinearities and nonvolatilities properties that will be shown in this work. Along with elements like resistors, capacitors and inductors, memristors are the fourth basic element that could be found in integrated circuits. So, they can be related to each other by the fundamental electrical equations, as shown in Figure 2.1.

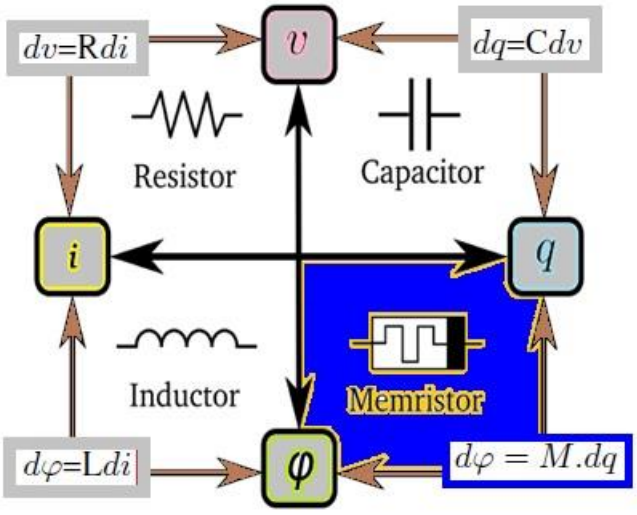

Figure 2.1 - The fundamental electrical passive elements adapted from [5]

These devices are defined as a two-terminal circuit element in which the flux between the terminals is a function of the amount of electric charge that passes through the device. They provide a functional relation between charge and flux, so they are said to be charge controlled if the relation between flux and charge is expressed as a function of electric charge or flux controlled if the relation between flux and charge is expressed as a function of the flux linkage [6].

In 2008 HP proposed the implementation of a memristor device using a very thin film of titanium dioxide between two platinum contacts which has two sides: one is doped with oxygen vacancies which is defined by a width w and the other is undoped which is defined by $(D - w)$ [4]. The oxygen vacancies on the doped side are positively charged ions, which makes them conductive. On the other hand, the undoped side has insulating properties. This device can be illustrated by the Figure 2.2.

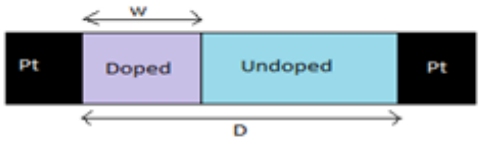

Figure 2.2 - Linear ion drift memristive device by HP Labs adapted from [5]

Therefore, when a positive voltage is applied, the positively charged oxygen vacancies on the doped side are pushed away and moved to the undoped side. This causes an increase in the percentage of the conducting and the size of doped region w to D size, due to the movement of the boundary between the materials. Therefore, the conductivity of the memristor increases and the resistance is lower, resulting in a change from R_{OFF} to R_{ON} [7].

When a negative voltage is applied, the positively charged oxygen vacancies that were moved to the undoped side are attracted and moved back from this side. This causes an increase of insulating titanium dioxide and the size of the undoped region increases, consequently the resistance is higher and increases the resistivity of the memristor. This results in a change of resistance from R_{ON} to R_{OFF} [7].

If no voltage is applied, the oxygen vacancies don't move, and the boundary stays in the same position it was on the last state. As such, the memristor memorizes the last voltage applied as well as its resistance value [5].

2.2 Linear Ion Drift Models

For the memristor created by HP Labs, linear ion drift models were proposed. To better understand these models, some equations will be needed. As such, a memristor device can be defined by eq. (1) and eq. (2), where V is the memristor voltage, i is the memristor current and $R(w,i)$ is the instantaneous resistance dependent on the state variable w of the device [5].

$$V = R(w,i)i \tag{1}$$

$$\frac{dw}{dt} = f(w,i) \tag{2}$$

Using the next fundamental known equations for current and voltage and replacing in eq. (1) the general equation for memristance (5) is obtained.

$$i = \frac{dq}{dt} \tag{3}$$

$$v = \frac{d\varphi}{dt} \tag{4}$$

$$M(q) = \frac{d\big(\varphi(q)\big)}{dq} \tag{5}$$

Having the system defined and applying the Ohm's Law relation, it is possible to obtain the simplified equation for memristance. The total resistance of the memristor is a sum of the resistance of the doped and undoped regions. These relations are defined by eq. (6) and eq. (7).

$$V(t) = R_{MEM}(w)i(t) \tag{6}$$

$$R_{MEM}(x) = R_{ON}(x) + R_{OFF}(1-x) \tag{7}$$

$$where\ \ x = \frac{w}{D}\ with\ \ x \in [0,1] \tag{8}$$

Inserting eq. (7) into eq. (6) and simplifying using the relation for the state variable the memristance system is defined by eq. (9).

$$V(t) = \left(R_{ON} \frac{w(t)}{D} + R_{OFF} \left(1 - \frac{w(t)}{D} \right) \right) i(t) \tag{9}$$

A linear ion drift model is defined by eq. (10) where μ_v is the dopant mobility, D is the limit of the boundary and R_{ON} is low resistance of the conducting side. Simplifying and replacing in eq. (9) results the equation for memristance system which for $R_{OFF} \gg R_{ON}$ leads to eq. (12).

$$\frac{dw(t)}{dt} = \mu_v \frac{R_{ON}}{D} i(t) \tag{10}$$

$$w(t) = \int \mu_v \frac{R_{ON}}{D} i(t) \ \ dt \iff w(t) = \mu_v \frac{R_{ON}}{D} q(t) \ \ dt \tag{11}$$

$$M(q) = R_{OFF} \left(1 - \frac{\mu_v R_{ON}}{D^2} q(t) \right) \tag{12}$$

The relation between the speed of the movement of the boundary between the doped and undoped regions and the current is given by eq. (13). Applying some window functions $f(w)$ described in the next chapters and multiplying by eq. (14) is possible to obtain a several new models for memristors [8].

$$\frac{dw(t)}{dt} = ki(t) \tag{13}$$

$$k = \frac{\mu_v R_{ON}}{D^2} \tag{14}$$

2.3 Window Functions

Applying a small voltage across the device will cause a very large electric field which can introduce considerable nonlinearities in ionic transport. That behavior can be observed at the external boundaries of the device. The boundary between the doped and undoped side stops, that is, the memristor is set to on or off state. This phenomenon is accounted for in the nonlinear ion dopant drift model behavior where none stimulus can switch back to the previous state [9].

This issue is reproduced using window functions, which are functions of state variables that can be used to control the boundaries between the two materials.

They can also introduce nonlinear behaviors close to these boundaries. There are several window functions that will be presented in this chapter.

2.3.1 Strukov Window Function

As mentioned before, the boundary restrictions can be modeled using the memristance system equation multiplied by the window function $f(w)$ intended as shown in eq. (15).

$$\frac{dw(t)}{dt} = \frac{\mu_v R_{ON}}{D} i(t) f(w) \tag{15}$$

The simplest window function is the Strukov window function [4]. This function is so basic that does not satisfy the boundary condition when $w = 0$ and $w = D$ [10]. The function is defined by eq. (16) and the issue mentioned before is defined by eq. (17).

$$f(w) = \frac{w(1-w)}{D} \tag{16}$$

$$\begin{cases} w \to 0 \\ w \to D \end{cases} \tag{17}$$

This means that the derivative of the state variable tends to zero, $\frac{dw}{dt} = 0$, and no external field can change the state. This window function also assumes that memristor remembers the amount of charge passing through the device remembering the position of the state boundary between the doped and undoped regions, and that is a problem, because in fact only remembers

the position of the state boundary between the two regions [9]. This window function is represented by Figure 2.3 for a normalized value of the width, $\frac{w}{D}$.

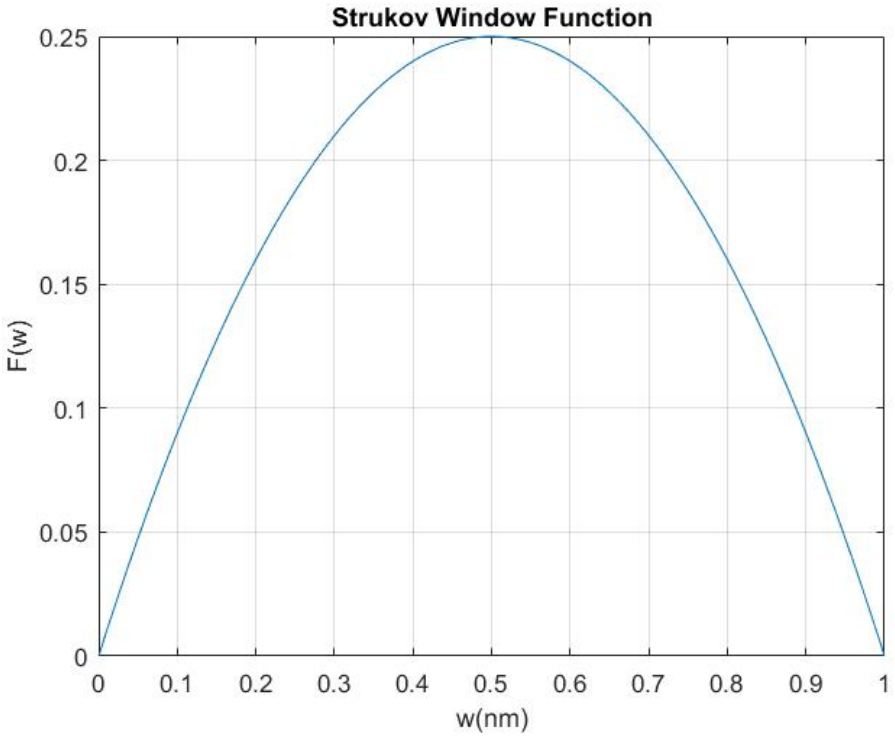

Figure 2.3 - Strukov Window Function

2.3.2 Joglekar Window Function

Joglekar proposed another window function [4]. In this case, the function introduced a new parameter. This parameter is a control parameter, p, which is a positive integer coefficient as represented in eq. (18). The memristor model will consider the product of eq. (13) by the function defined by eq. (18).

$$f(x) = 1 - (2x - 1)^{2p} \tag{18}$$

The control parameter controls the linearity of the memristor model and shows that the function becomes more rectangular as it increases. So, the function becomes more linear as p increases and this variation is shown in Figure 2.4 considering the normalized width, $\frac{w}{D}$. To optimize this window function, the values for p can vary from 0 to 10, ensuring zero drift at the boundaries [5].

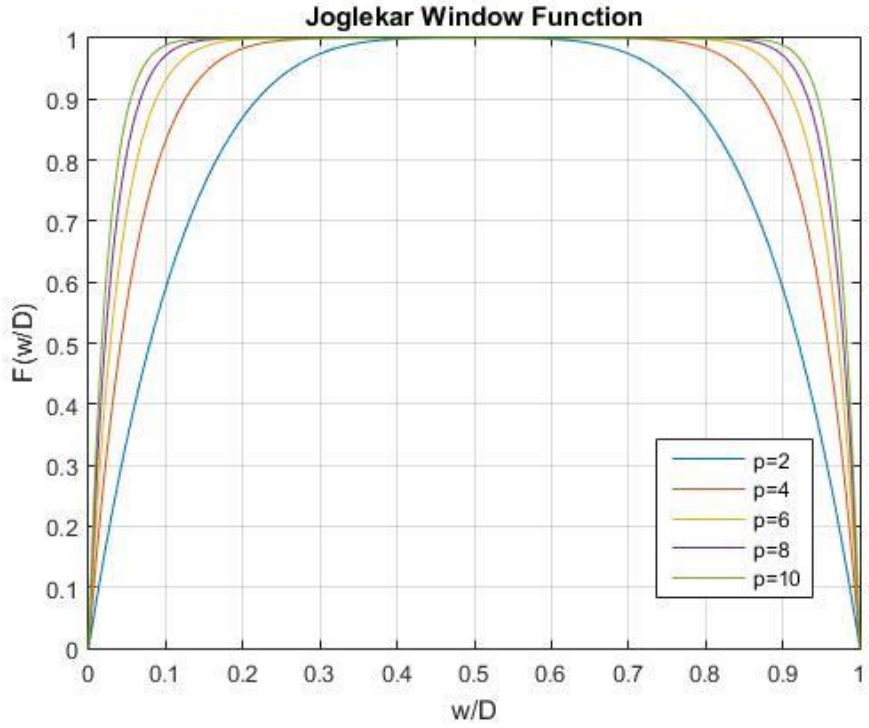

Figure 2.4 - Joglekar Window Function

Although presenting more accurate results, this window function contradicts the operation principle of the memristors. Once the limits of the boundaries are reached, e.g., when the width of the device hits 0 or D, the model cannot adjust the state of the device. Therefore, the memristor cannot come back from the state it had before even when a reversed bias voltage is applied.

2.3.3 Biolek Window Function

To overcome the problem about the terminal state observed by the previous window functions, Biolek proposed another solution [4]. This window function not only considers the changes in the state variable but also the total charge passing through the memristor device. The window function proposed by Biolek is eq. (19).

$$f(x) = 1 - [x - stp(-i)]^{2p} \tag{19}$$

In this window function, the $stp(-i)$ is a step function of current which is the control parameter defined by the limits presented in eq. (20). This step function resolves the restriction problem of the nonvolatility of the memristor presented in the previous model.

$$stp(i) = \begin{cases} 1, & i \geq 0 \\ 0, & i < 0 \end{cases} \tag{19}$$

This one allows the change in the value at terminal state from minimum to maximum when is applied reverse bias voltage. The behavior of this function in Figure 2.5 for a normalized value of $\frac{w}{D}$.

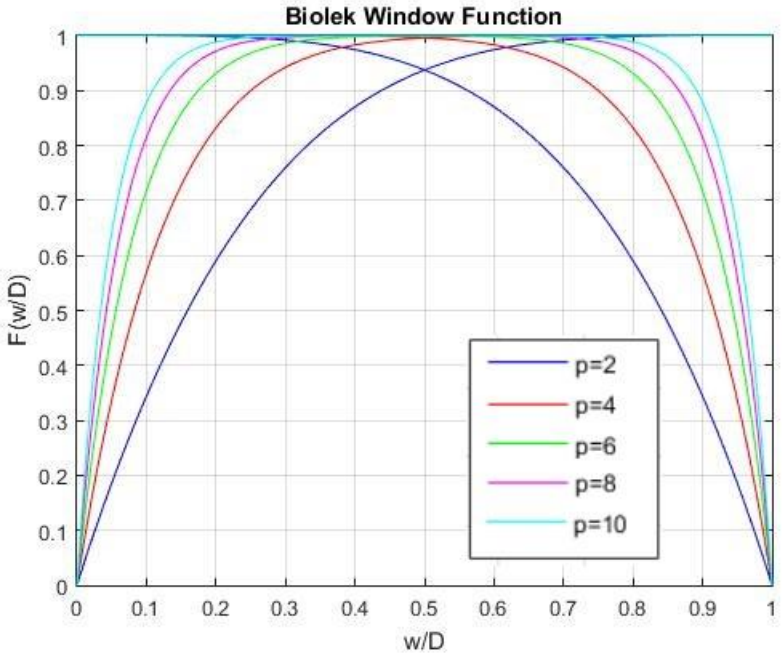

Figure 2.5 - Biolek Window Function

However, Biolek function shows problems about the nonlinear properties of memristors. They are not clearly observed and there is no continuity of function at the boundaries [5].

2.3.4 Prodromakis Window Function

To overcome Joglekar window function state problem that restricts an important characteristic that memristors have, Prodromakis suggested a new window function defined by eq. (21). With this window function, a new control parameter j appears to adjust the maximum value that the window can reach. In addition, introducing $+1$ outside the brackets ensures that $f_{max}(x) = 1$ and varying the control parameter p, it allows the window function to scale upward between a range $0 \leq f_{max}(x) \leq 1$ [11].

$$f(x) = j(1 - [(x - 0.5)^2 + 0.75]^p) \tag{21}$$

This parameter p optimizes the effect of nonlinearities at the boundaries like Joglekar window function. Unlike the models proposed by Joglekar and Biolek, the parameter p can take any positive real number which offers a greater extent of flexibility [11].

The variation of parameters p and j are shown in Figure 2.6 and Figure 2.7 respectively.

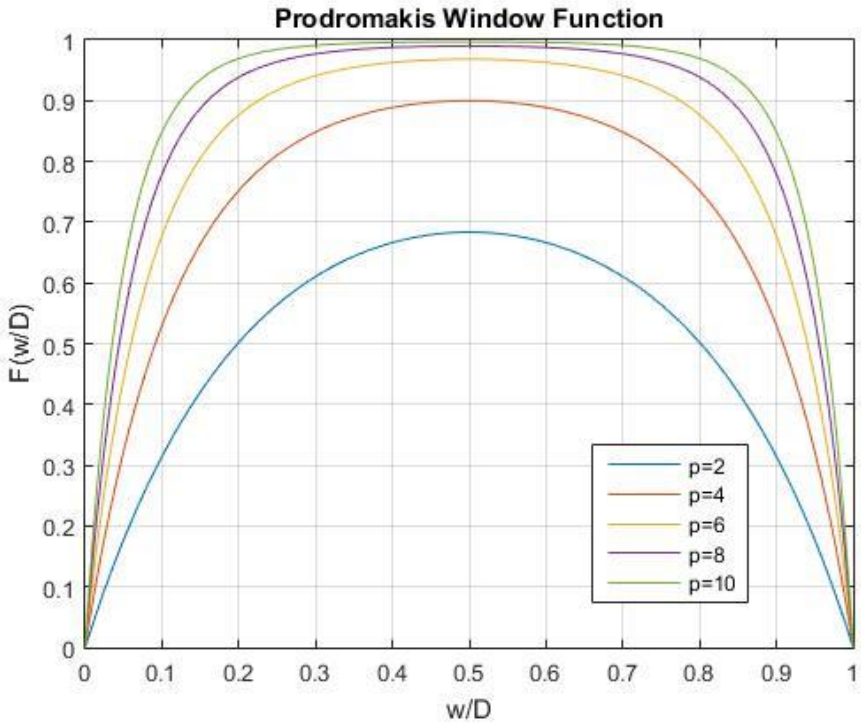

Figure 2.6 - Prodromakis Window Function with p as variable

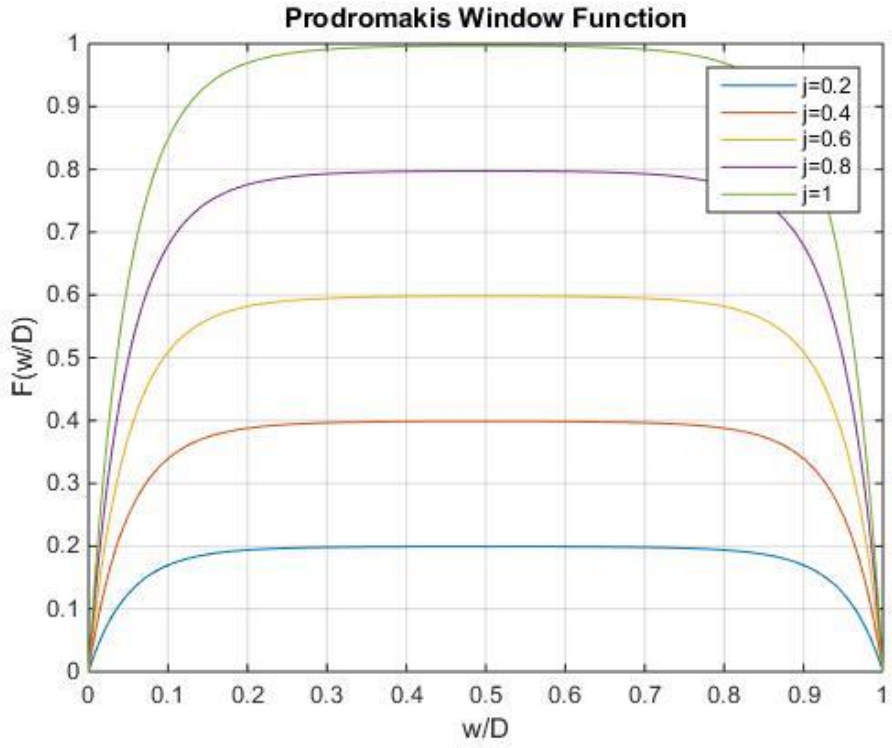

Figure 2.7 - Prodromakis Window Function with p=10 and j as variable

2.4 NonLinear Ion Drift Models

Nonlinear ion drift models assume that a voltage controlled memristor has a nonlinear dependency between the voltage and the internal state derivative, which can be expressed by eq. (22), where a and m are constants, m an odd integer and $f(w)$ is the window function.

$$\frac{dw}{dt} = af(w)v(t)^m \tag{22}$$

This shows an asymmetric switching behavior. Over the years, memristors have be proven to show nonlinear behavior which make the models considered in this subsection more appropriated to mimic the behavior of these devices. The relationship between the current and voltage is defined by eq. (23).

$$i(t) = w(t)^n \beta sinh\big(\alpha v(t)\big) + \chi\big[e^{\gamma v(t)} - 1\big] \tag{23}$$

The fitting parameters α, β,γ and χ are experimental and n determines the influence of the state variable w on the current, which is normalized between $[0,1]$.

The operation of this model assumes an ON and OFF state. When the ON state is active, the state variable w is close to the upper limit of the interval and the current is expressed by $\beta sinh(\alpha v(t))$, which describes a tunneling effect. When the OFF state is active, the state variable w is close to lower limit of the interval and the current is expressed by $\chi[e^{\gamma v(t)} - 1]$, which describes the behavior of an ideal diode [12].

2.4.1 Simmons Tunnel Barrier Model

Instead of the previous models that had two resistors in series, a resistor in series with an electron tunnel barrier is considered in this model, as shown in Figure 2.8.

In this nonlinear model, memristor show nonlinear ion drift effects and asymmetric switching behavior due to an exponential dependence of the movement of the ionized dopants, e.g., changes in the state variable. The state variable in Simmons Tunnel Barrier is the width x of the tunnel [12].

The tunneling effect causes the memristor to force the oxygen vacancies through a tunnel with width x, by introducing a resistor with high resistance in the doped side. The control mechanism of this model is current, therefore it is widely used in the digital applications such as flash memories [13]. The relationship between voltage and current is described by the complex expression shown in eq. (24).

$$i(t) = \tilde{A}(x, v_g)\phi_1(v_g, x)e^{\left(-B(v_g,x)\phi_1(v_g,x)\right)^{0.5}} - \tilde{A}(x, v_g)(\phi_1(v_g, x) + e|v_g|)e^{\left(-B(v_g,x)(\phi_1(v_g,x)+ev_g)\right)^{0.5}} \quad (24)$$

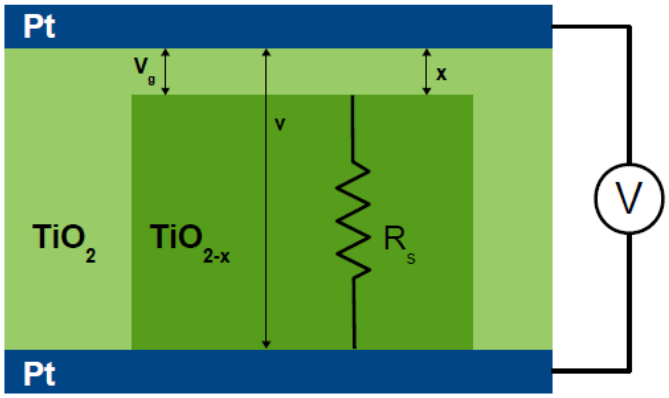

Figure 2.8 - Memristor structure based on Simmons Tunnel Barrier; adapted from [9]

As mentioned before, the state variable x represents the width of this model, and differentiating this variable, is obtained the equation for drift velocity of oxygen vacancies described by eq. (25). The fitting parameters are C_{on}, C_{off}, a_{off},a_{on}, i_{off}, i_{on} and b where C_{on} is much larger than C_{off} on the magnitude of the change of state variable x. The currents i_{off} and i_{on} represent the negative and positive threshold with a_{on} and a_{off} forcing the lower and the upper bound for x, respectively [9].

Once the range is defined, the derivative expression of the state variable is much smaller than the state variable itself which makes a big advantage because there is no need for window functions.

$$\frac{dx}{dt} = \begin{cases} C_{off}.sinh\left(\frac{i}{i_{off}}\right).e^{\left[-e^{\left(\frac{x-a_{off}}{w_c}-\frac{|i|}{b}\right)}-\frac{x}{w_c}\right]} & if\ i > 0 \\ C_{on}.sinh\left(\frac{i}{i_{on}}\right).e^{\left[-e^{\left(\frac{x-a_{on}}{w_c}-\frac{|i|}{b}\right)}-\frac{x}{w_c}\right]} & if\ i < 0 \end{cases} \quad (25)$$

This is the first model accounting for the fact that drift and diffusion are in opposite direction for an applied positive voltage whereas for an applied negative voltage it is in same direction.

The Simmons tunnel barrier is the most accurate model of memristor. However, this model is not fully compatible with all types of memristors due to the ambiguous nature of the relationship between current and voltage [5].

The complexity in its implementation and the none explicit relation between the voltage and current (I-V relationship), makes it necessary to develop new models with simpler mathematical functions while preserving the accuracy of this one [9].

2.4.2 ThrEshold Adaptive Memristor Model (TEAM)

This Threshold Adaptive Memristor Model (TEAM) is a simplified of Simmons Tunnel Barrier model representing the same physical model with much simpler expressions as shown in eq. (26), where k_{off}, k_{on}, α_{off} and α_{on} are constants [12].

$$\frac{dx}{dt} = \begin{cases} k_{off}.\left(\frac{i(t)}{i_{off}} - 1\right)^{\alpha_{off}}.f_{off}(x), & i > i_{off} \\ 0, & i_{on} < i < i_{off} \\ k_{on}.\left(\frac{i(t)}{i_{on}} - 1\right)^{\alpha_{on}}.f_{on}(x), & i < i_{on} \end{cases} \tag{26}$$

The introduction of current threshold concept with separated window function for ON and OFF state represented by eq. (27) and eq. (28) makes the I-V relationship clearly observed. These functions represent the dependence on the sate variable x and force bounds x since $f_{off}(x)$ is used when $\frac{dx}{dt}$ is positive and is zero around a_{on} and $f_{on}(x)$ is used when $\frac{dx}{dt}$ is negative and is zero around a_{off}. This shows an asymmetric behavior like in Simmons Tunnel Barrier model.

$$f_{on}(x) = e^{-e^{\left(\frac{x-a_{on}}{w_c}\right)}} \tag{27}$$

$$f_{off}(x) = e^{-e^{\left(\frac{x-a_{off}}{w_c}\right)}} \tag{28}$$

The explicit behavior of the relationship between the voltage and current is expressed by eq. (29), which assuming that his characteristics are similar to the linear ion drift model, the memristance changes linearly in x [9]. Assuming the relationship used for Simmons Tunnel Barrier, the memristance is represented by eq. (30) which is modeled by an exponential behavior with the fitting parameter λ described by eq. (31).

$$v(t) = \left[R_{on} + \frac{R_{OFF} - R_{ON}}{x_{OFF} - x_{ON}}.(x - x_{on})\right] i(t) \tag{29}$$

$$v(t) = R_{on}.e^{\left(\frac{\lambda}{x_{off} - x_{on}}\right)(x - x_{on})}.i(t) \tag{30}$$

2.4.3 Polynomial Model

This model is based on the first definition of memristor introduced by professor Chua and matches with HP and TEAM models, expressed by a simple polynomial representation of state equation [14]. However, a physical implementation is not achieved. This equation is modeled by Taylor series expansion shown in eq. (32) and its behavior shows a dependency on current.

$$\frac{dx}{dt} = \sum_{n=0}^{N} a_n . i^n + \sum_{m=0}^{M} b_m . w^m . i + \sum_{p=1}^{P} \left(c_p . i + d_p\right)^p \left(e_p . w + f_p\right)^p \tag{31}$$

2.5 Conclusion

In this chapter several memristor models proposed in the literature were presented. These models pertain to two main groups, e.g., the linear and the nonlinear ion drift models.

The need for using window functions to account for the memristor behavior near the borders was duly justified. The main limitations of the several models were presented.

3. MatLab Implementation of Memristor Models

The main objective of this chapter is to present the development of the memristor models described in the previous chapter. For a first validation of the models implemented in MatLab, the LTspice implementation of macromodels for each of the presented linear models was considered.

The section 3.1 describes the macromodels developed in LTspice as well as the respective window functions. The section 3.2 describes the implementation in MatLab and validation of these models, and at last, in section 3.3, the nonlinear models are presented.

Finally, the use of Runge-Kutta method for the implementation of the models, is considered as a way of minimizing numerical errors in the evaluation of the differential equations.

Conclusions on the benefits of using variable step algorithm are driven.

3.1 LTspice Macromodels

The memristor macromodel was created based on the mathematical model of the HP Labs memristor. The symbol was created using the memristor code in the Appendix A and then introduced in the circuit shown in Figure 3.1. The circuit is composed by a sinusoidal voltage source with an amplitude of $1.2V$ and a frequency of $1Hz$, considering phase 0.

The values for parameters as well as the simulation conditions were based in [4]. The initial values used were $R_{ON} = 100\Omega$, $R_{OFF} = 16k\Omega$, $R_{INIT} = 11k\Omega$, $D = 10nm$ and $\mu_V = 10f$ for $T = 1s$.

The window functions are implemented in the memristor code in the Appendix A too. By uncommenting in the respective window function, it is possible to simulate the memristor behavior as shown in the next sub-sections.

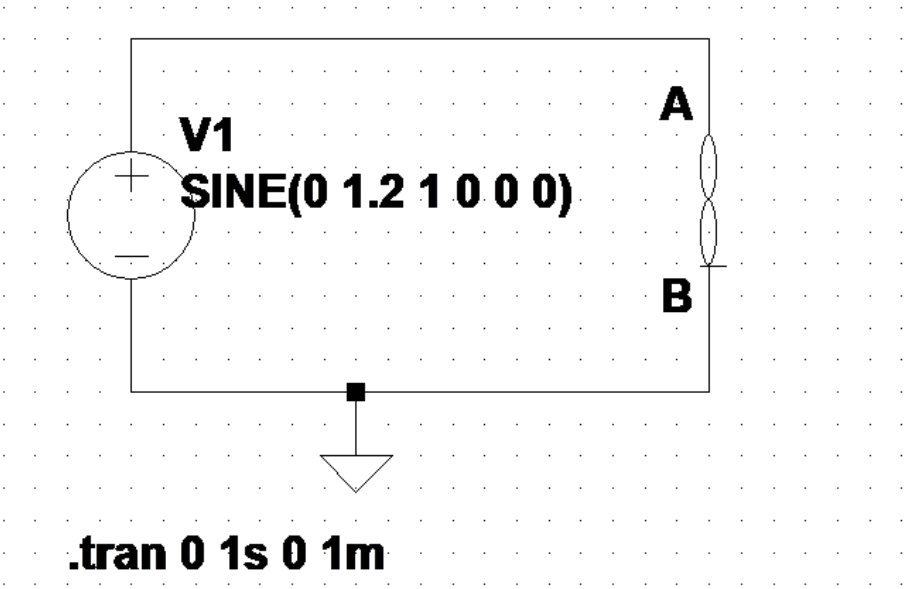

Figure 3.1 - LTspice Macromodel

3.1.1 Strukov Window Function

The first window function applied was the Strukov window function. The simulation was based on a transient analysis of the implemented macromodel for $D = 10nm$. The results demonstrate the variation of current and voltage and I-V hysteresis loop.

Observing Figure 3.2, the current of the memristor I_{MEM} varies up to approximately $100\,\mu A$ for an applied voltage of $1.2\,V$. Applying Ohm's Law, the total resistance of the memristor, R_{MEM}, show that the values are limited between $11k\Omega$ and approximately $12\,k\Omega$ as shown in Figure 3.3.

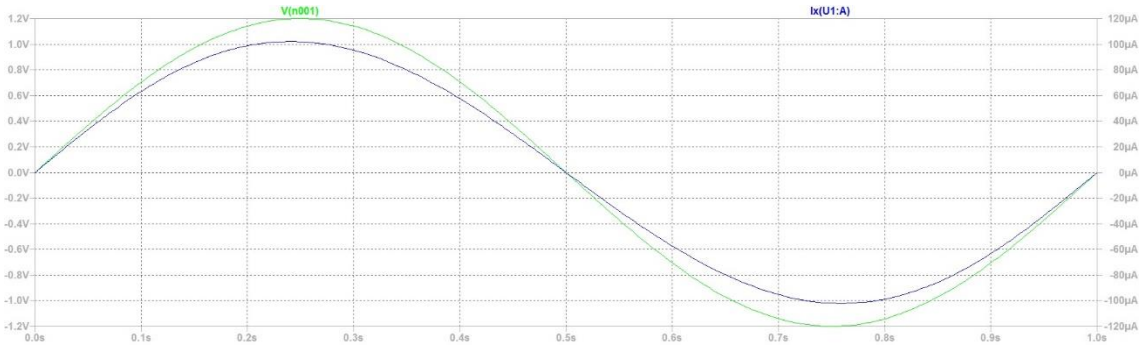

Figure 3.2 - Strukov memristor model

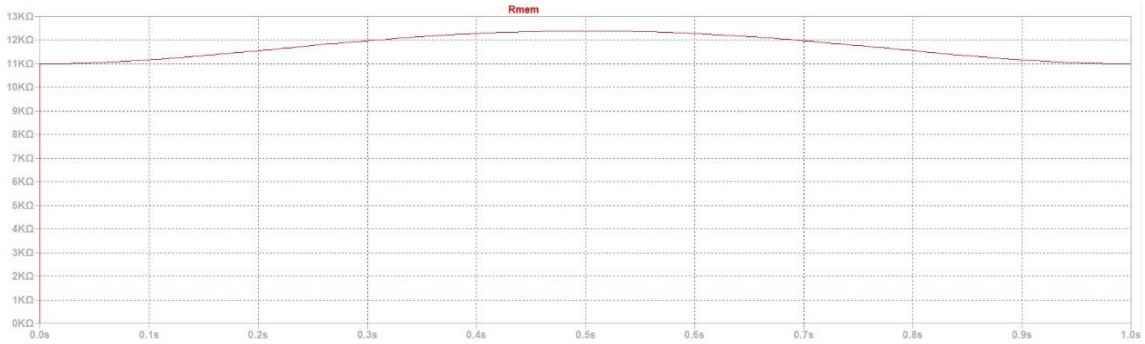

Figure 3.3 - Variation of the Resistance in memristor model with Strukov Window Function

When a voltage is applied, the memristor only gives a small variation for its memristance showing that the full range of the memristor width is not used. This window function does not have flexibility to control the nonlinearities at the boundaries and that is proven by the hysteresis loop in Figure 3.4.

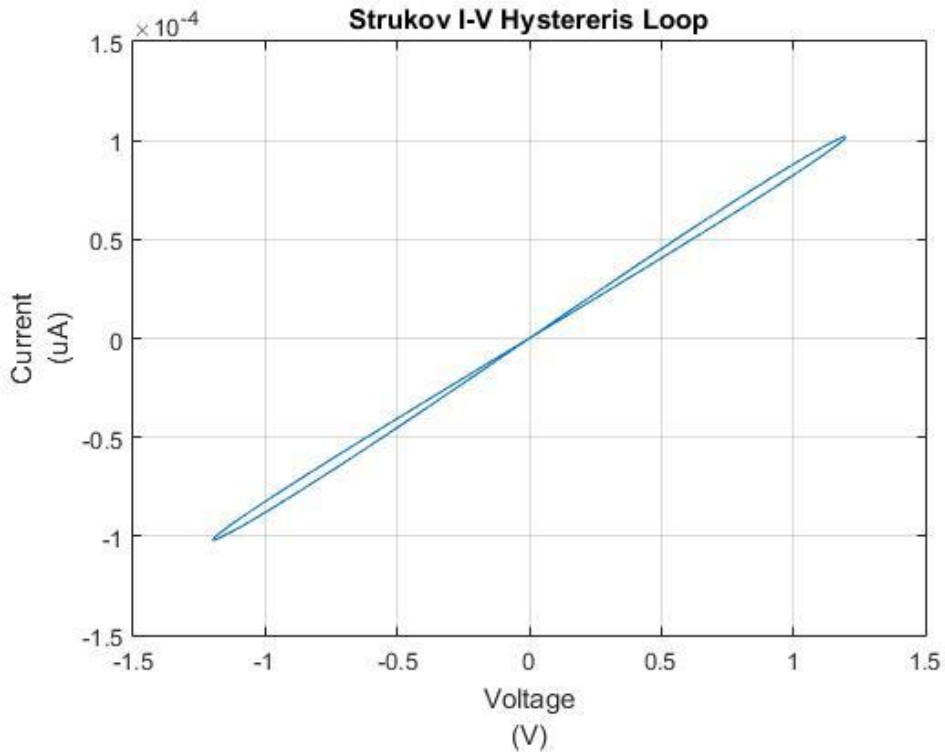

Figure 3.4 - Strukov Hysteresis Loop

3.1.2 Joglekar Window Function

From the eq. (18), mentioned in chapter 2, as the value of p increases, the effects of nonlinearities are more evident. The course of the current and the voltage over the time is shown in Figure 3.5 and Figure 3.6.

In this window function, with $p = 5$, the current I_{MEM} varies up to approximately $300\ \mu A$ for an applied voltage of $1.2V$.

The resistance R_{MEM} gives a full range of values for the memristor between nearly from $0k\Omega$ to $11k\Omega$ which can prove that this model reaches all the memristor width as shown in Figure 3.7.

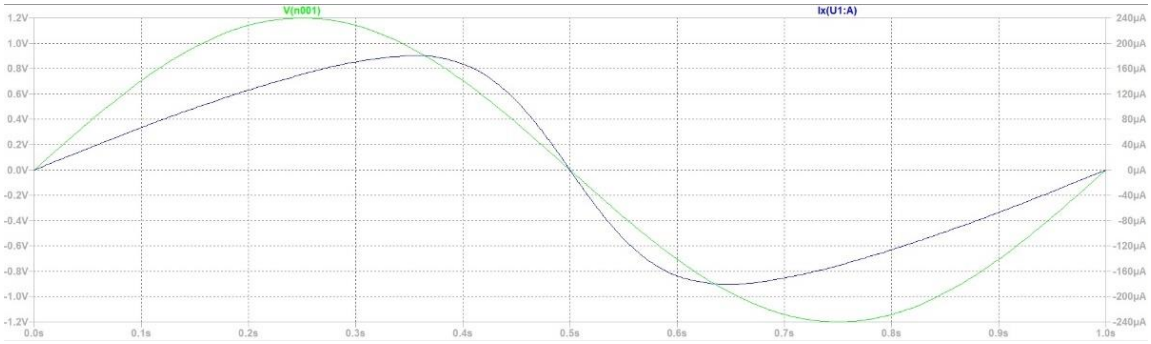

Figure 3.5 - Joglekar memristor model with p=1

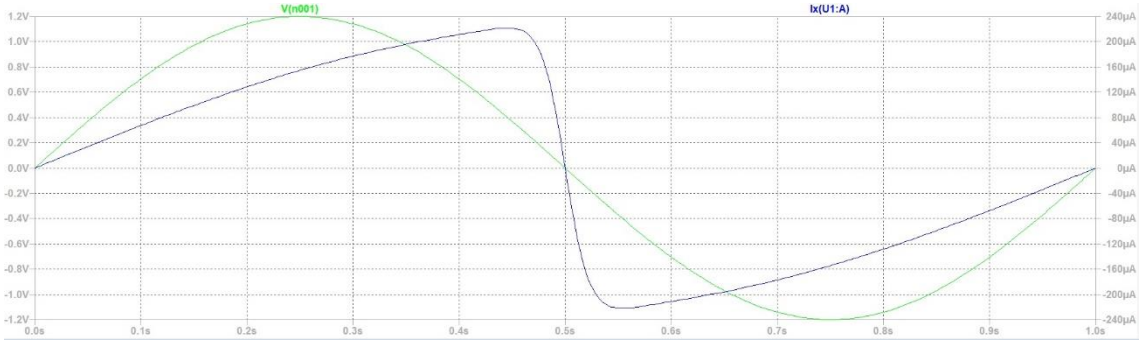

Figure 3.6 - Joglekar memristor model with p=5

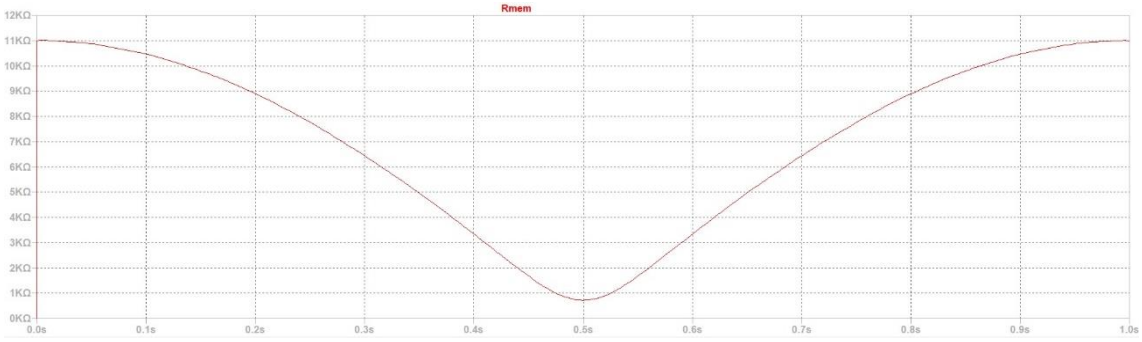

Figure 3.7 - Variation of the Resistance in memristor model with Joglekar Window Function

In comparison with Strukov window function, this one shows a better ability to preserve the deformation by a stimulus as shown in Figure 3.8. Increasing the parameter p causes the loop to be wider.

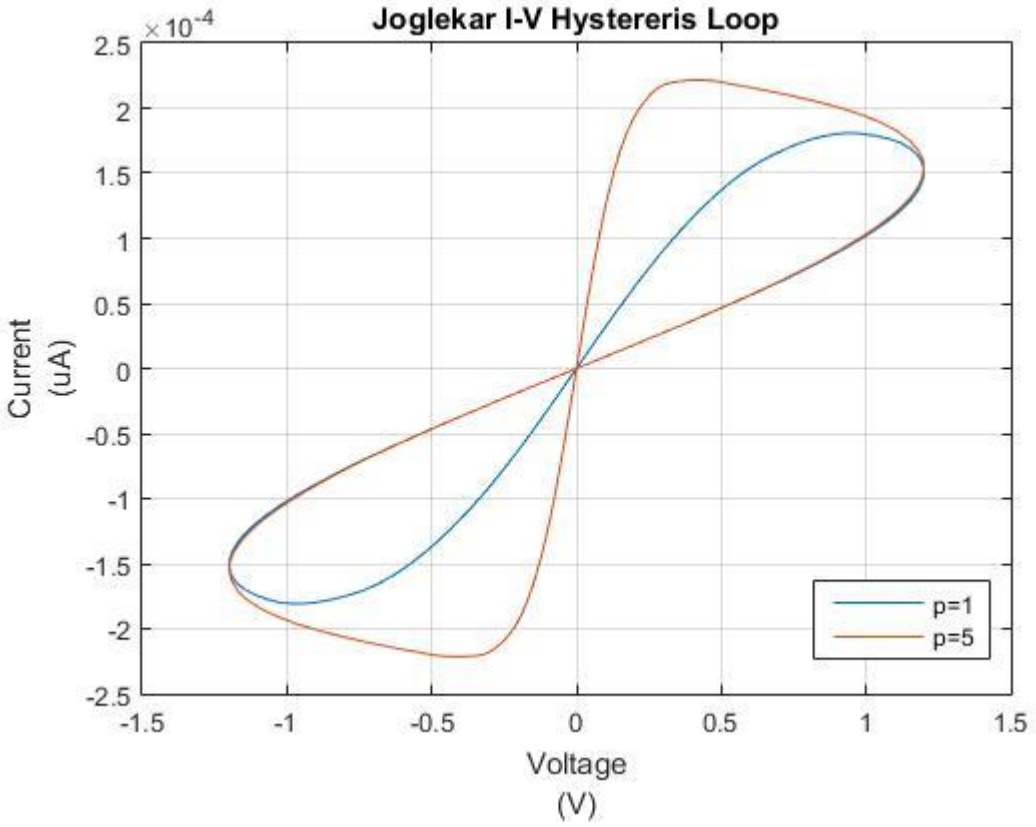

Figure 3.8 - Joglekar Hysteresis Loop comparison

3.1.3 Prodromakis Window Function

Figure 3.9 shows the variation of current and voltage for a parameter $p = 1$, where a value of $120\mu A$ is achieved. Otherwise Figure 3.10 shows that current I_{MEM} varies close to values of $150\mu A$ for an applied voltage of $1.2V$. The increase of parameter p, in eq. (21) from chapter 2, causes the course between current and voltage to be delayed.

The resistance R_{MEM} takes values between $5k\Omega$ to $11k\Omega$, shown in Figure 3.11, where the full range of the memristor width is not used.

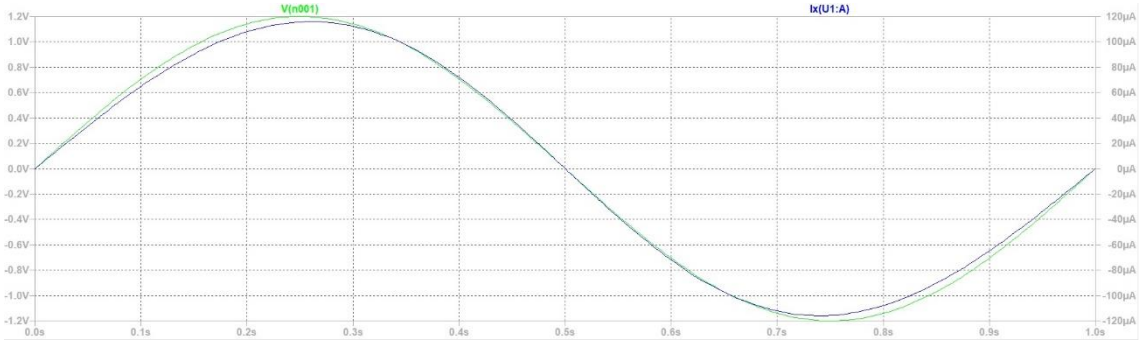

Figure 3.9 - Prodromakis memristor model with p=1

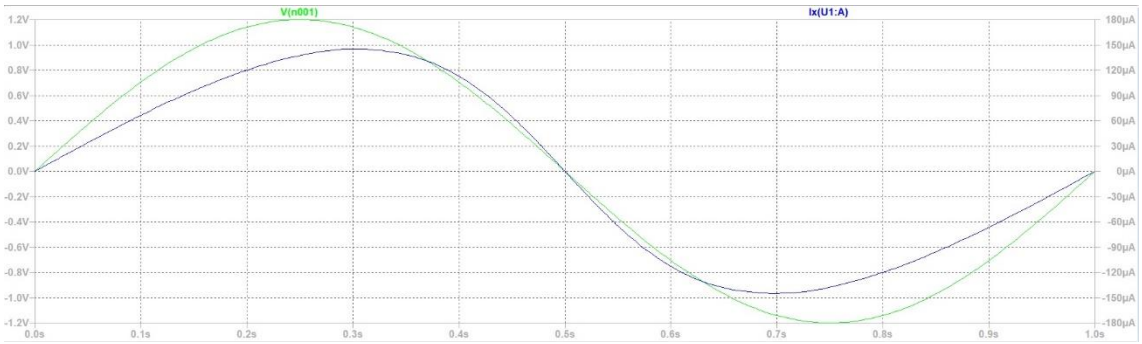

Figure 3.10 - Prodromakis memristor model with p=5

Figure 3.11 - Variation of the Resistance in memristor model with Prodromakis Window Function

The hysteresis loop in Figure 3.12 is shown to be asymmetrical while the OFF state of the device is highly nonlinear compared with another. The changes are only considered in state variables and not in the charges.

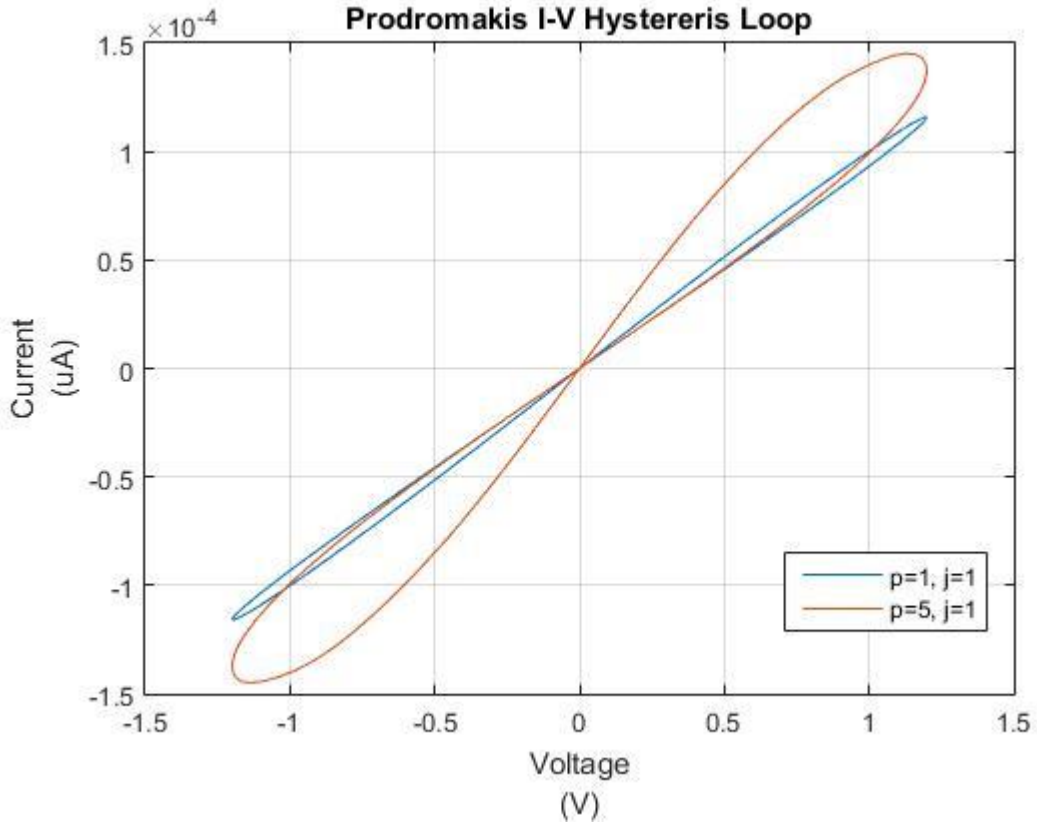

Figure 3.12 - Prodromakis Hysteresis Loop comparison

3.1.4 Biolek Window Function

In Figure 3.13 a current of $150\mu A$ is achieved, whit a parameter $p = 1$. In Figure 3.14, it is observed that is possible to reach values for the current I_{MEM} approximately as $200\mu A$ for the same applied voltage in all simulations of $1.2V$. Using the Ohm's Law, the resistance R_{MEM} have wider ranges varying between $1k\Omega$ to $11k\Omega$ shown in Figure 3.15, which considers practically all the range of the memristor width.

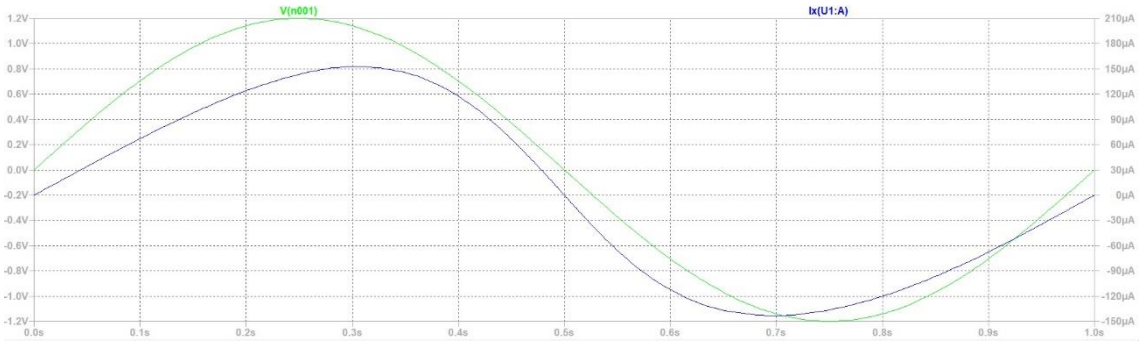

Figure 3.13 - Biolek memristor model with p=1

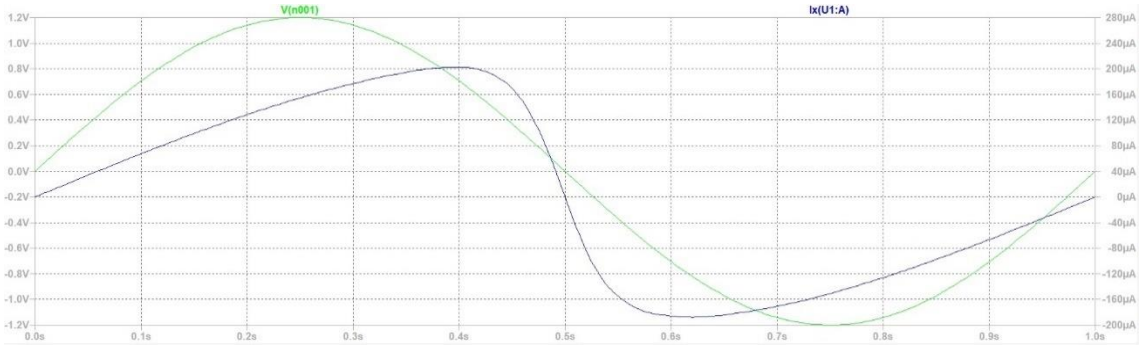

Figure 3.14 - Biolek memristor model with p=5

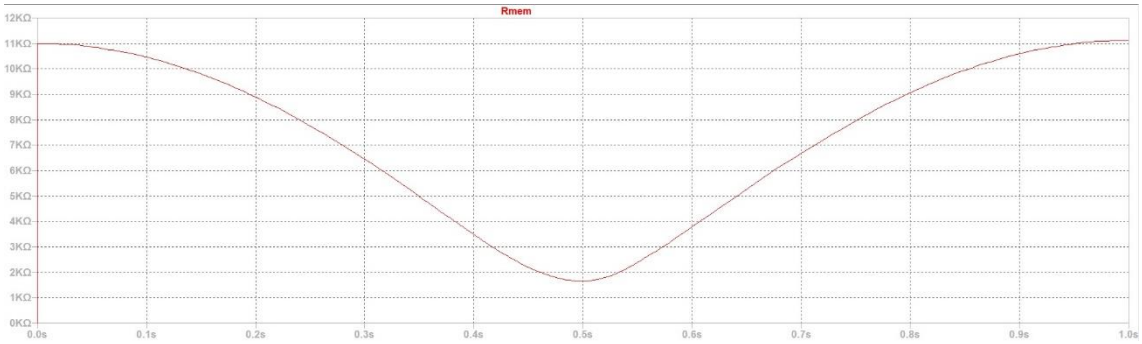

Figure 3.15 - Variation of the Resistance in memristor model with Biolek Window Function

The Figure 3.16 show the I-V hysteresis loop comparison. The results are very similarly compared with Prodromakis window function where the effect of increasing the parameter p is evident.

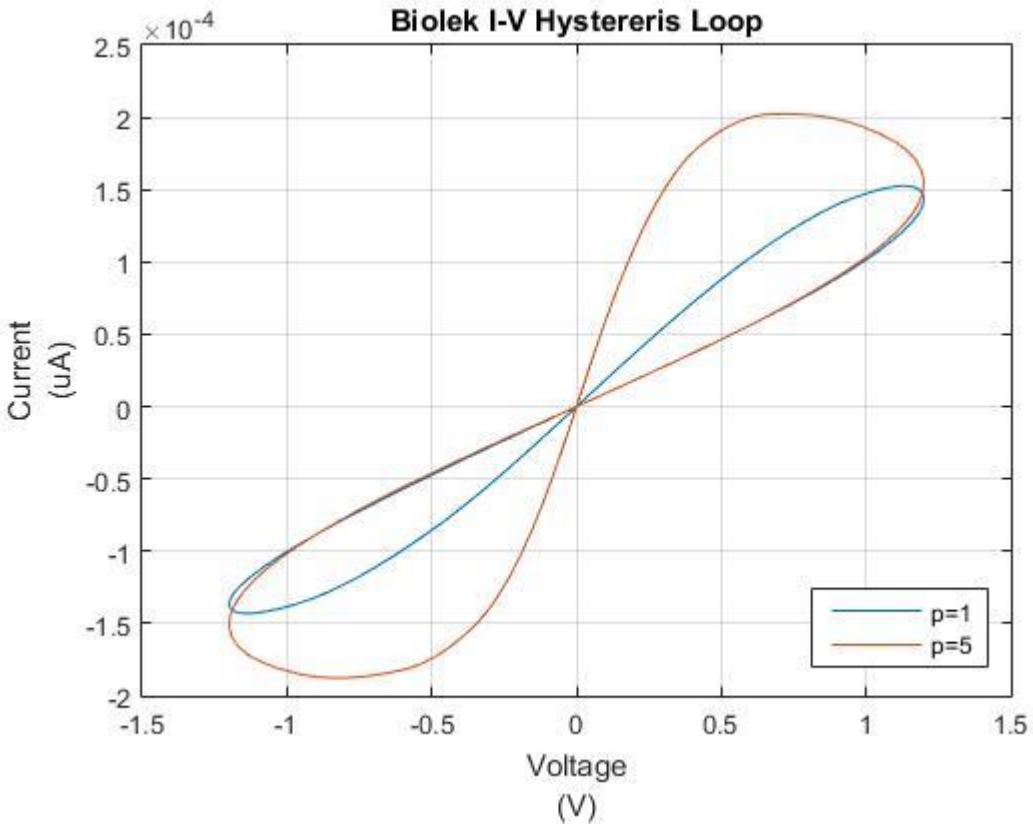

Figure 3.16 - Biolek Hysteresis Loop comparison

3.1.5 Conclusion

Figure 3.17 shows the I-V hysteresis loop for all the models described before under the same parameters to see the differences between each one. The Strukov window function shows visible limitations in terms of resistance and that is big issue to represent a memristor model.

The Biolek and Prodromakis window functions are very similar in terms of the resistance values reached, and so do why, their hysteresis loops are very identical.

The Joglekar window function seems to have the best results. That is verified by the ability of a system to preserve a deformation, e.g., the hysteresis loop when a voltage is applied. When a voltage is applied, this window function can achieve higher currents compared to others.

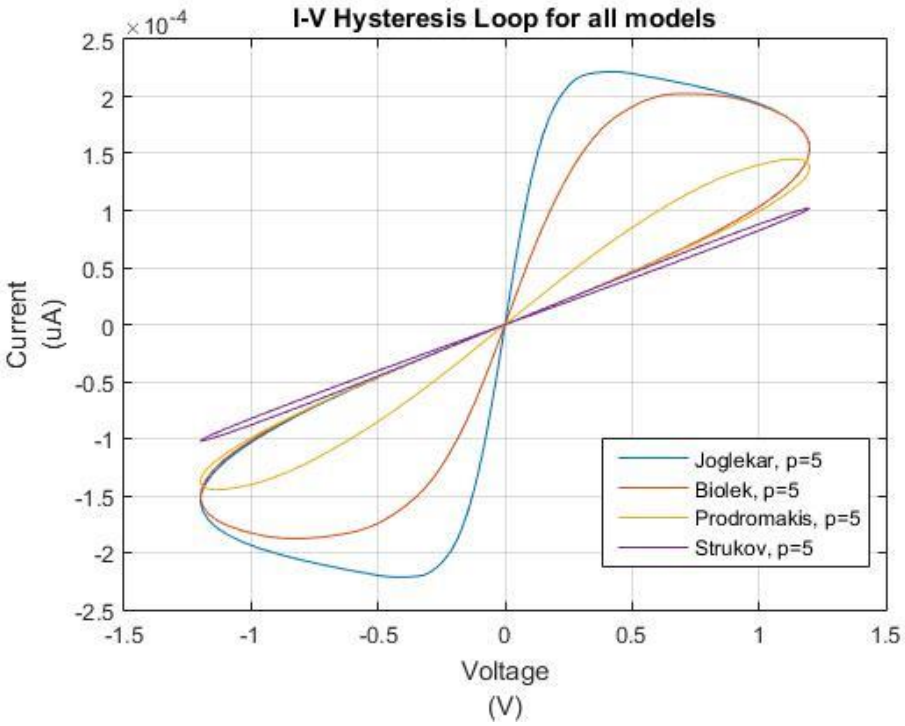

Figure 3.17 - I-V Hysteresis Loop comparison for all models

3.2 MatLab Implementation

This section considers the implementation of the linear and nonlinear models in MatLab. After a first implementation in LTspice, the objective was to validate the previous implemented models under the same conditions. The nonlinear models were not implemented in LTspice because of their non-linearities characteristics which are not clearly observed in this software.

The simulations in MatLab were made for an input voltage of $1.2V$ considering only one cycle, to avoid numerical approximation errors. The parameters used were $R_{ON} = 100\Omega$, $R_{OFF} = 16k\Omega$, $R_{INIT} = 11k\Omega$, $D = 10nm$ and $\mu_V = 10f$.

The code used for this validation is shown in Appendix B. By selecting the required function, it is possible to obtain a comparison between the two implementations.

3.2.1 Strukov Window Function

In Figure 3.18 it is possible to observe the comparison between the two implementations. The function represented by the blue color was obtained according to the implementation in MatLab while the function represented by the red color was obtained according to the implementation in LTspice.

This model does not depend on the value of p and so do why the hysteresis loop depend on the reachable values of resistance that the memristor can achieve, which are in a very small range. Variation on the achieved values may be due to numerical approximation errors.

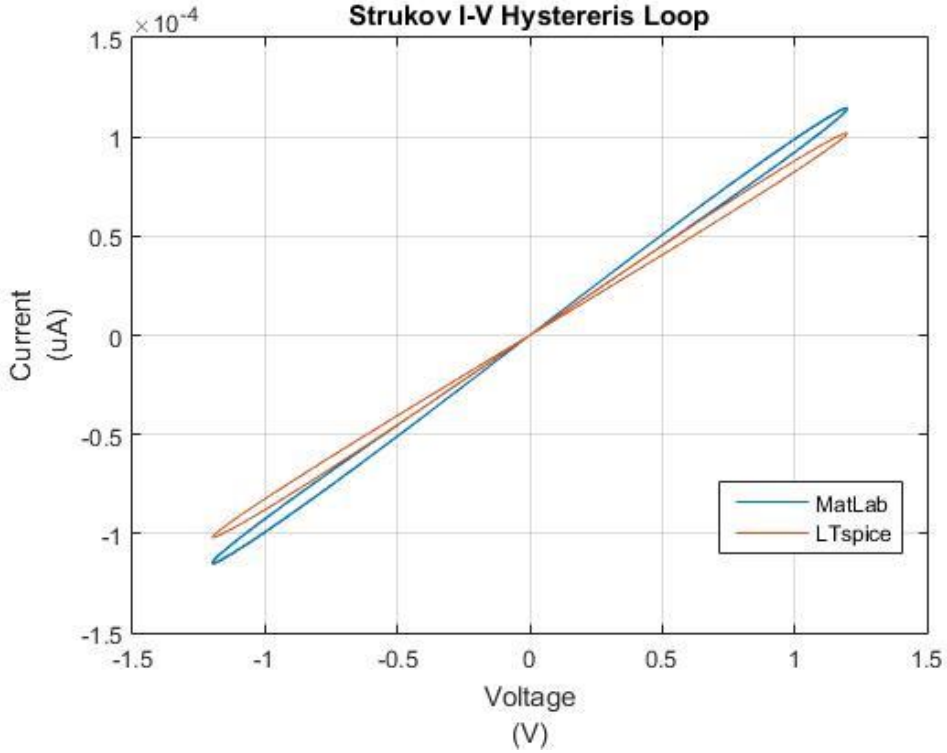

Figure 3.18 - Hysteresis Loop comparison for the Strukov Window Function

3.2.2 Joglekar Window Function

In Figure 3.19 it is possible to observe the comparison of both implementations with $p = 5$. In this window function it is possible to achieve higher currents compared to others. The switching behaviour is much more sensitive on voltage compared to the previous window which validate the higher current values. The hysteresis loop is wider compared to the other models since the full range of the memristor is reachable.

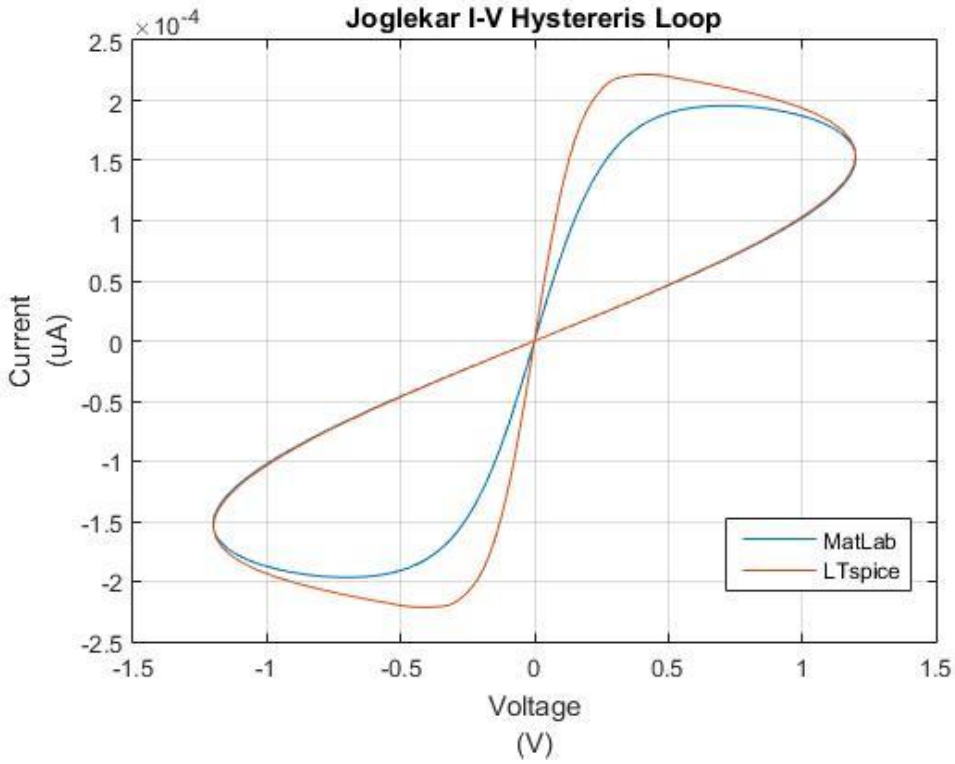

Figure 3.19 - Hysteresis Loop comparison for the Joglekar Window Function

3.2.3 Prodromakis Window Function

In this implementation was considered the parameters $p = 5$ and $j = 1$, which are the best values, as seen in the previous chapter, to approximate this window function to the required behaviour model. The two implementations are very similar. This behaviour is shown in Figure 3.20.

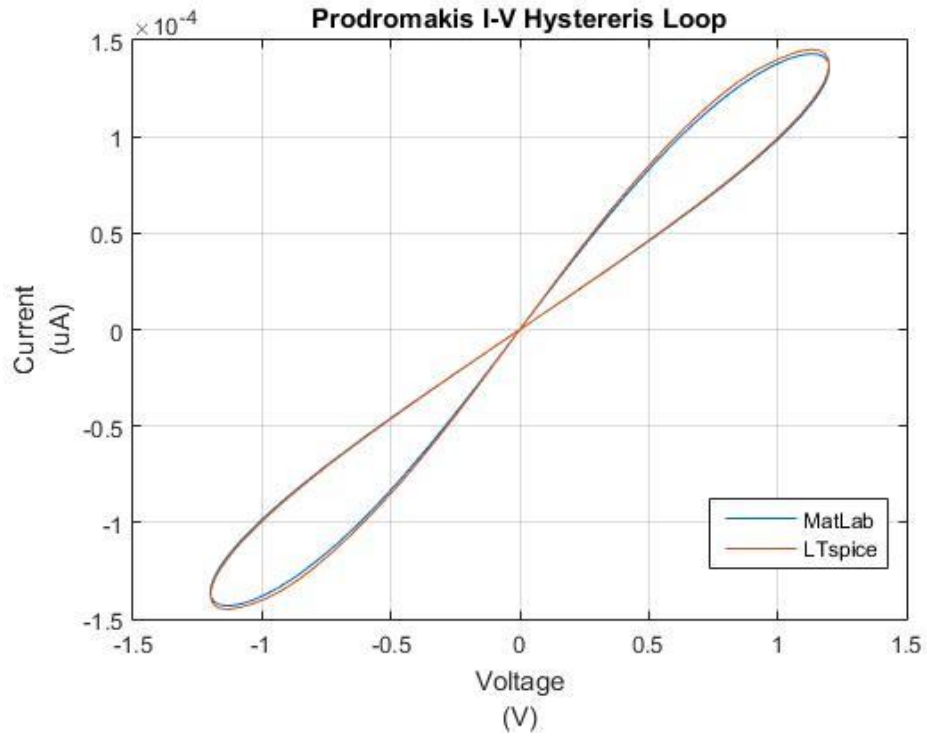

Figure 3.20 - Hysteresis Loop comparison for the Prodromakis Window Function

3.2.4 Biolek Window Function

The Figure 3.21 shows the comparison for the two implementations with $p = 5$. Once again, there are slight variations in current values. The ideal symmetrical loop is not clearly observed and so the non-linearity behaviour too. However, the hysteresis loop implemented by MatLab is similar to the one implemented by LTspice.

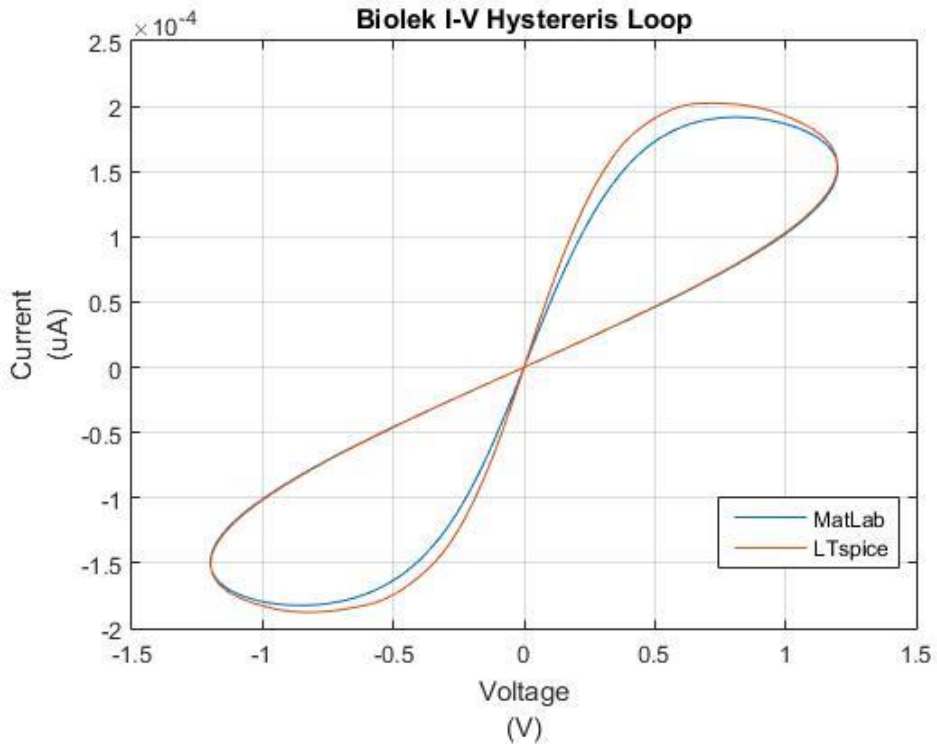

Figure 3.21 - Hysteresis Loop comparison for the Biolek Window Function

3.2.5 ThrEshold Adaptive Memristor Model (TEAM)

As mentioned in the beginning of this chapter, the nonlinear models were not implemented in LTspice due to their characteristics. So, considering only the implementation in MatLab, the hysteresis loop for this model is presented in the next figures. The used code is shown in Appendix C.

The simulations were made with an input current of $8mA$ using just one cycle with $f = 1kHz$. The parameters $R_{ON} = 50\Omega$, $R_{OFF} = 1k\Omega$ and the initial state of 0.2 were used.

Assuming the current-voltage relation with a behaviour like (29), the memristance changes linearly and the result is shown in Figure 3.22.

Assuming the Simmons Tunnel Barrier current-voltage relation, given by (30), the memristance changes exponentially due to any change in the tunnel barrier width, as shown in

The two hysteresis loops represent the desired asymmetric behaviour with switching OFF slower than switching ON [12].

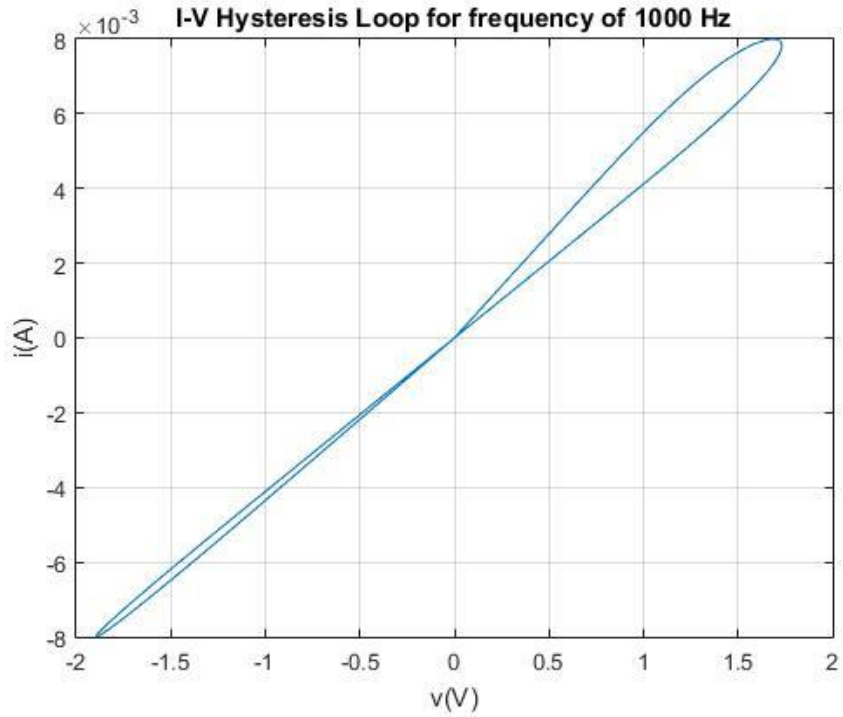

Figure 3.22 - TEAM I-V Hysteresis Loop with linear Memristance

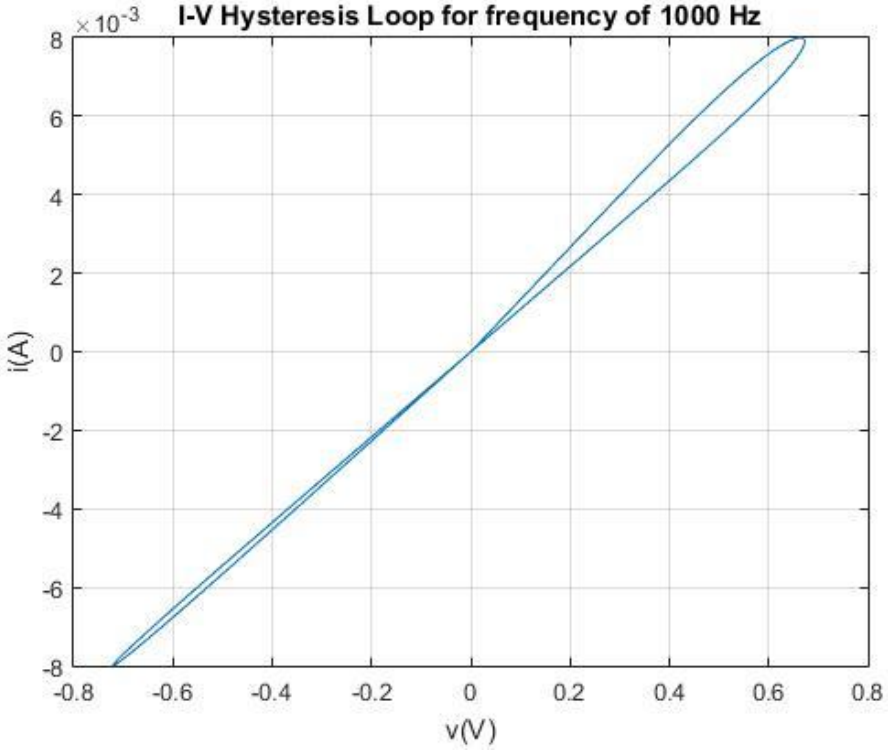

Figure 3.23 - TEAM I-V Hysteresis Loop with exponential Memristance

3.2.6 Conclusion

The previous implemented models in Matlab were based on the Euler method for the resolution of the differential equation. Fixing the step to 0.0025, with a frequency of $1Hz$, and considering several periods, it is possible to show that the various loops do not coincide, due to the accumulation of numerical approximation errors as illustrated in Figure 3.24.

To overcome this limitation two different approaches may be considered. In the first one, smaller time steps may be considered. This will compromise the time efficiency of the model. A second approach considers the use of variable step algorithm. In this work, the Runge-Kutta method was implemented as described in the next section.

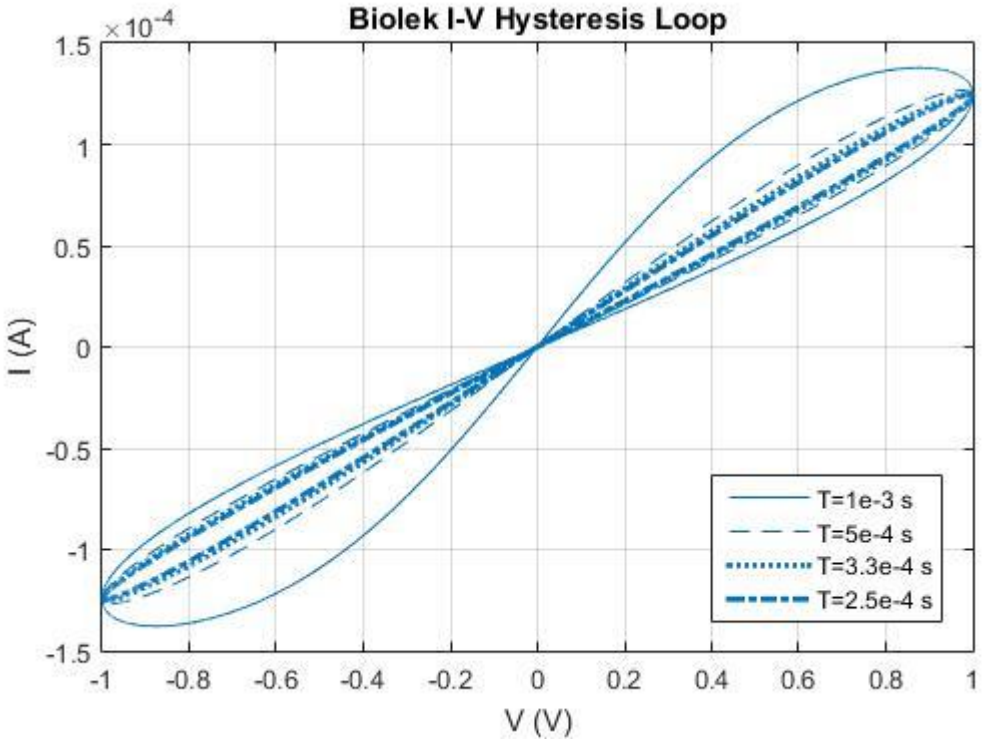

Figure 3.24 - Variation of the Euler Method

3.3 Application of Runge-Kutta Method to the implemented models

In this section, the application of Runge-Kutta method for the evaluation of the memristor models is presented. Sub-section 3.3.1 describes, in a brief introduction, how this method works as well as its characteristics. In sub-section 3.3.2 the results for each linear implemented model are presented.

Finally, the conclusions are driven, showing a comparison between this method and the Euler method, presented in the previous chapters.

3.3.1 Introduction

The Runge-Kutta is a method of approximation of numerical analysis to solve ordinary differential equations. The one used in this chapter, is the Runge-Kutta 4.5. Instead of the Euler method, this one uses an adaptive step size. This means that the step can be adapted during the implementation for each model [15].

So, the method is defined by an initial step, h, and an initial variable, ε, which defines the initial error. Initializing with a moderate step size, the expected error is compared with the value of ε. If the expected error is larger than ε, the step size is reduced, and the current step is recalculated. If the expected error is smaller than ε, the current step keeps the same and is slightly enlarged in the next iteration [16].

Basically, the value of ε is obtained by the calculation between a Runge-Kutta of order 4 and a Runge-Kutta of order 5. The next flowchart, in Figure 3.25, shows the algorithm of this method. The results for each memristor model are shown in the next section.

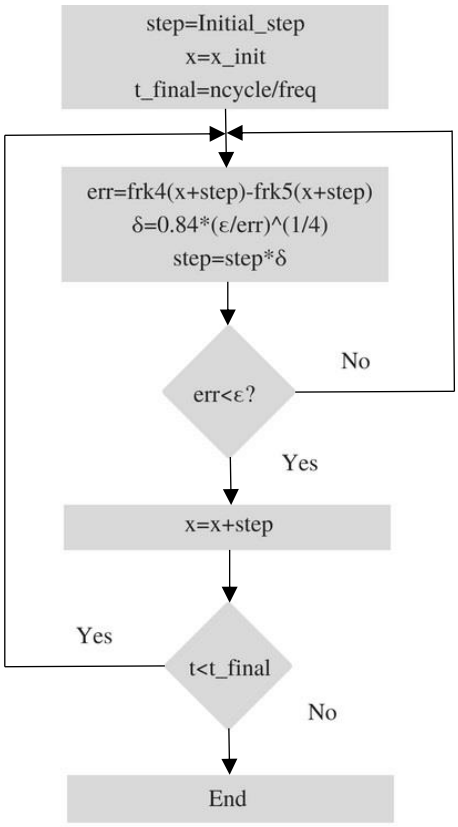

Figure 3.25 - Flowchart of the Runge-Kutta Method

3.3.2 Implemented Models with Runge-Kutta Method

The simulations presented in this section were based on the MatLab script language attached in Appendix D. The simulations were made under the same conditions with the same parameters as Euler Method. The parameters used were $p = 5$, $\mu_V = 10f$, $D = 10\,nm$, $R_{ON} = 100\Omega$, $R_{OFF} = 16\,k\Omega$ for an initial state of 0.5. The parameter ε is specifically used in this method with a value of $1e^{-17}$.

The results for each memristor model are shown in Figure 3.26. The comparison between the two methods is obtained through the attached code in Appendix E. The use of Biolek window function was merely a choice just to demonstrate that with less points than Euler method, the Runge-Kutta method shows similar results.

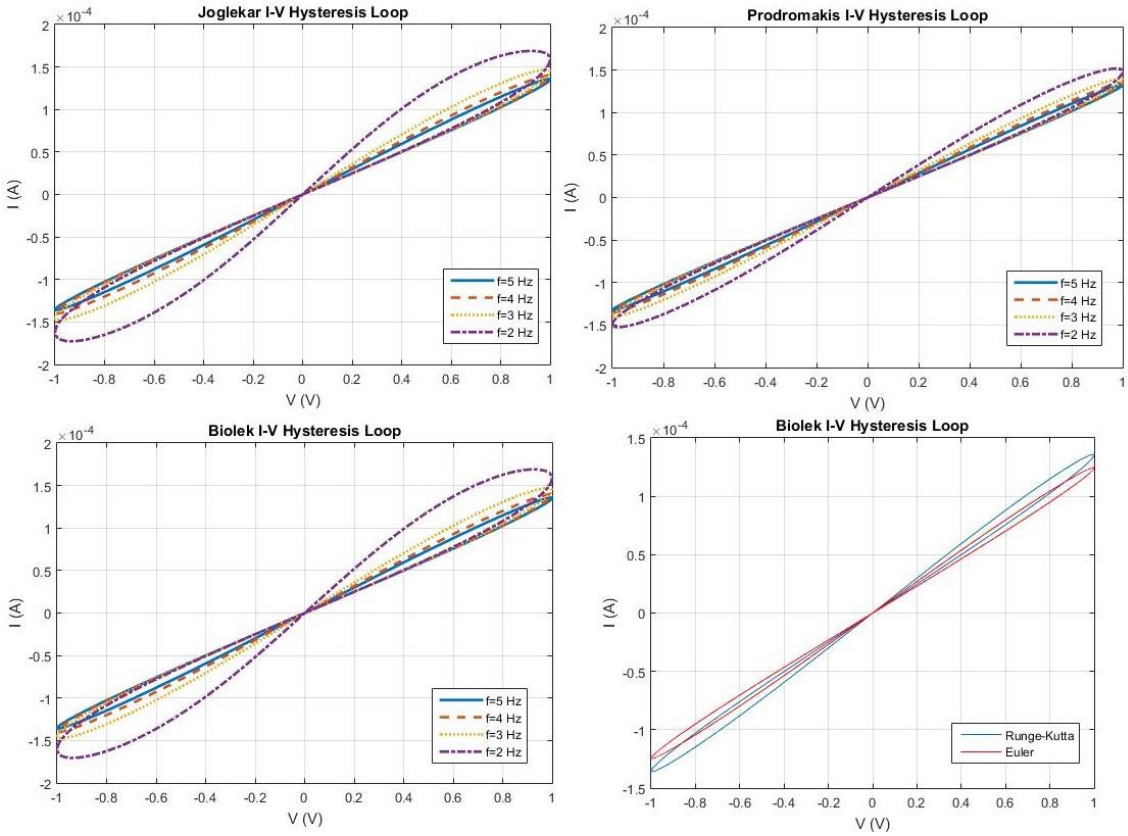

Figure 3.26 - Implemented models with Runge-Kutta Method

3.3.3 Conclusion

The results obtained for all models show the accuracy of this method proving the results shown in the previous sections by the Euler method.

As the initial error decreases, less points are needed and so the propagation of the numerical error is more evident.

The Runge-Kutta method shows a better efficiency due to the fact of using fewer points and being called less times during its computation. For example, with a frequency of $5Hz$, the Runge-Kutta method only needs 88 points instead of the 10001 points that Euler method needs.

The comparison between the two methods, for each implemented model, are shown in the Table 1. Euler>f and Runge-Kutta>f are functions referred to the implementation of each window function.

Table 1 - Comparison between Euler and Runge-Kutta Methods

MatLab Functions	Joglekar Window Function	Prodromakis Window Function	Biolek Window Function	Npoints with f=5Hz
Euler>f	0.875 s	0.930 s	0.953 s	10001
Euler	1.348 s	1.391 s	1.432 s	10001
Runge-Kutta>f	0.015 s	0.016 s	0.023 s	88
Runge-Kutta	0.132 s	0.112 s	0.117 s	88

4. VerilogA Memristor Models

4.1 Introduction

In this chapter the implementation of memristor models in VerilogA is considered, as a way of enabling the simulation of memristor-based circuits in the Cadence environment. Section 4.1 describes the existing types of oscillators and their applications. Section 4.2 addresses the main challenges in the implementation in VerilogA of the memristor models and discusses the approaches existent in the literature. The adopted approach and results obtained are presented.

Section 4.3 introduces types of memristor-based oscillators. Then for the oscillator topology considered, the analytical characterization of their behavior is presented, and the oscillation characteristics are driven.

Finally, section 4.4 presents the simulation results and the conclusions are driven.

4.2 VerilogA Memristor Models

The implementation in VerilogA of memristor models offers a more efficient way of performing the electric simulation of circuits using these devices. As previously described in chapter 2, several models have been proposed, where the relation between the voltage and current is characterised by a differential equation.

For the development of the memristor model in VerilogA, three different approaches are usually adopted. In the first approach the internal unknown, i.e. w or x, are declared as *real variables*, and the differential equation is modelled using idt function [9]. However, the way *real variables* are treated inside differential equations is not guaranteed to offer consistent results in different VerilogA compilers. In the second approach the differential relationship is modelled by coding time integration inside, using Euler approximation. In this case, the model has to be synchronized with the simulation time, i.e., either through the *abstime* function or by imposing the time step in the model and then fixing the step in the transient simulation [17].

Moreover, in transient simulation, this approach inhibits the possibility for taking advantage of the simulator facilities, e.g., convergence aiding techniques, truncation error estimation or timestep control. In the last case, the differential equation is modelled using the Kirchoff's current law at an additional internal node *nx*, as illustrated in Figure 4.1 [18].

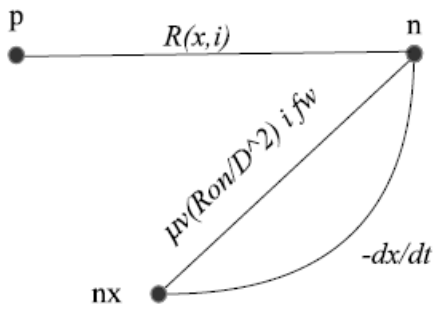

Figure 4.1 - Graphical representation of the approach for implementing the memristor model in VerilogA

As illustrated in listing 1, the internal variable x is considered as the voltage between node nx and the output node n. In the memristor model implementation a *distance* nature is defined as a way of enabling viewing the value for the internal variable at port *n_position.* It is to be noted that to prevent numerical errors the memristor dimensions are considered in nanometers. The VerilogA code for the window functions is represented in listing 2. In the particular case for the Biolek function a smoothing function, i.e., *smoothstep,* is to avoid the hard discontinuity of the step function to raise convergence issues during transient simulation. Finally, the code for the smoothing functions used is represented in Appendix F.

```
    `include "constants.vams"
    `include "disciplines.vams"
    nature distance
        access=Metr;
        units= "";
        abstol=0.01n;
    endnature
    discipline Distance
        potential distance;
    end
module MemristorModel( p,n, x_pos);
inout p,n;
inout x_pos;
electrical p,n,nx;
Distance x_pos;
parameter real D=10; // in nm
parameter real miuv=1e-5;// *1e-9 because D in nm
parameter real Ron=1e2;
parameter real Roff=16e3;
parameter real window=0; //0-no window; 1=Joglekar 2=Biolek  3=Prodromakis
parameter real j=1;// for Prodromakis
parameter real p_coef=5;
parameter real sm=1e-8 from (0:inf); // for smoothing function
parameter real Gmin=1e-12 ; // for solving homotopy issues
parameter real R_ini=11e3;
parameter real maxslope=1e15 from (0:inf);  //for transient analysis convergence
parameter real Kclip=50 from (0:inf); //for transient analysis convergence
parameter real xinit=0.2 from (0:1);

real x;    // contains w/D
real y;
real MR ;   // memristor resistance
real f1,f2,fw1,fw2,clip0,clip1;
real fw;   // window factor
real Ix;
real init;
analog begin
    x=V(bx1);
    @(timer(0.0))
        x=xinit;
    y=smoothclip(x-Roff/(Ron-Roff),sm)+Roff/(Ron-Roff);    //avoid division by zero
    Metr(x_pos)<+y;
    MR=Ron*(y)+Roff*(1-y);
    Metr(x_pos)<+MR;
    f1=V(p,n)/MR;   //contains current
    I(p,n)<+f1+Gmin*V(p,n);
    Ix=I(p,n); // for biolek function
    case (window)
        1:fw= Joglekar(x,p_coef);
        2:fw=Biolek (x,p_coef,Ix,m);
        3:fw=Prodromakis(x,p_coef,j);
    default fw=1;
    endcase
    f2=miuv*1e9*Ron/(D*D)*f1*fw;
    fw1=smoothstep(0-x,sm);
    fw2=smoothstep(x-1,sm);
    clip0=(safeexp(Kclip*(0-x),maxslope)-f2)*fw1;
    clip1=(-safeexp(Kclip*(x-1),maxslope)-f2)*fw2;
    I(bx1)<+ddt(-x);
    I(bx1)<+f2+clip0+clip1;
end
endmodule
```

Listing 1: Memristor model implementation in VerilogA

```
analog function real Joglekar;
     real x,p1; input x,p1;
     Joglekar=(1-pow(2*x-1,2*p1));
endfunction

analog function real Biolek;
     real x,p1,Ix,m; input x,p1,Ix,m;// m is for the modified Biolek
   Biolek=(1-pow((x-smoothstep(-Ix,sm)),2*p1) +m*pow(sin(x*`M_PI),2))/(1+m);
endfunction

analog function real Prodromakis;
     real x,p1,j; input x,p1,j;
     Prodromakis=j*(1-pow((pow(x-0.5,2)+0.75),p1));
endfunction

branch (nx,n) bx1;
```

Listing 2: VerilogA code for the implementation of the window functions.

4.2.1 Memristor model Simulation Results

For the simulation of the memristor model, the behavior of a memristor with the parameters represented in Table 2 is considered. The transient simulation results for the Joglekar, Biolek and Prodromakis window functions were obtained with the schematic represented in Figure 4.2, where the frequency of operation to be considered may be selected through variable *f* and the amplitude of the voltage is given by the value of variable *amp*.

Table 2 - Memristor Parameters

R_{on}	R_{off}	D	μ_v
0.1kΩ	38kΩ	10nm	10e-5$m^2 s^{-1} V^{-1}$

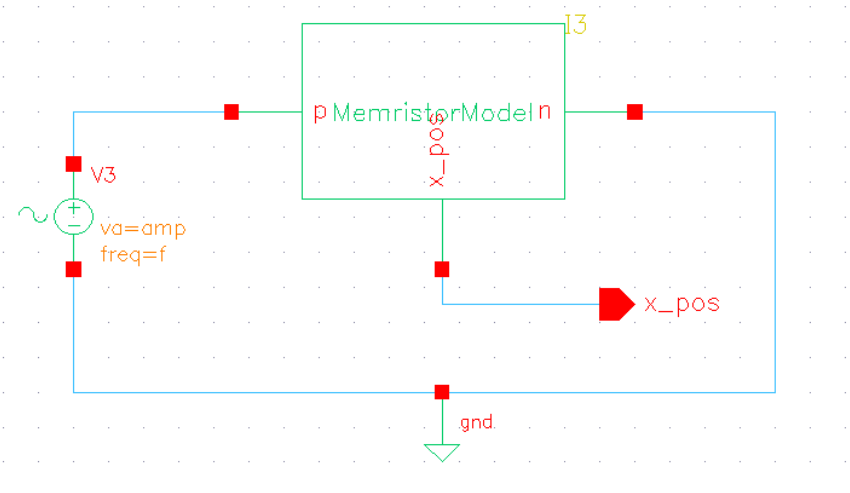

Figure 4.2 - Schematic of the circuit for transient simulation of memristor

A: Joglekar Window Function

For the particular case of Joglekar window function, the *window* variable is set to 1. In the first case, a frequency of $1Hz$, and a voltage of $200mV$ of amplitude were considered. In Figure 4.3 the transient response for the current, voltage and x_position are illustrated, whereas in Figure 4.4 the hysteresis loop is represented.

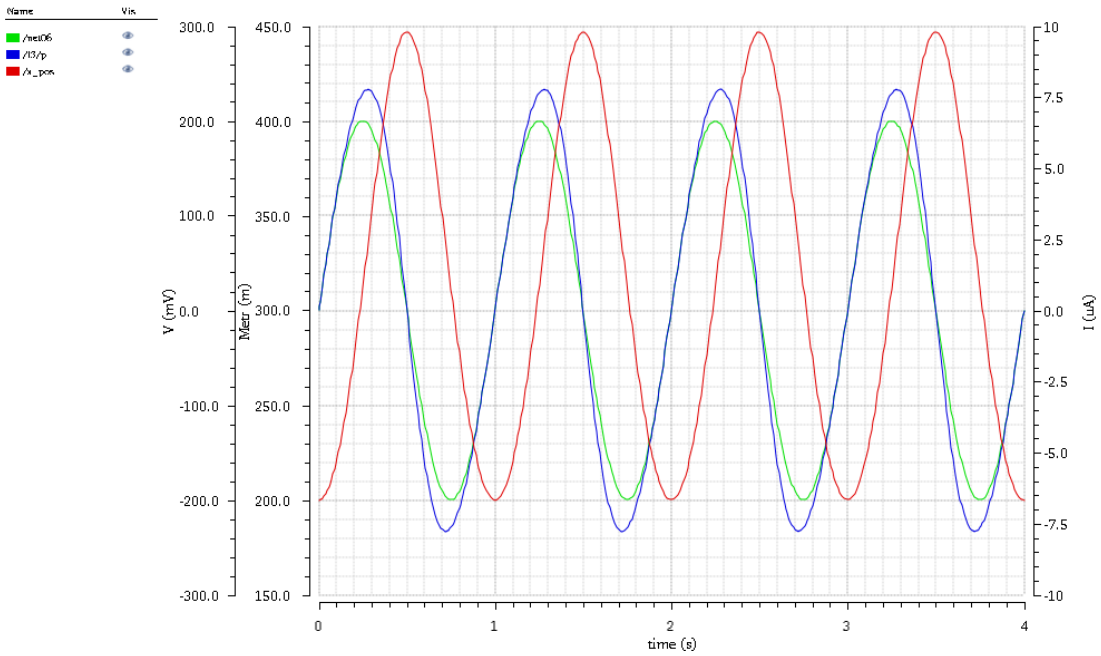

Figure 4.3 - Transient simulation of current, voltage and x_position with f=1Hz

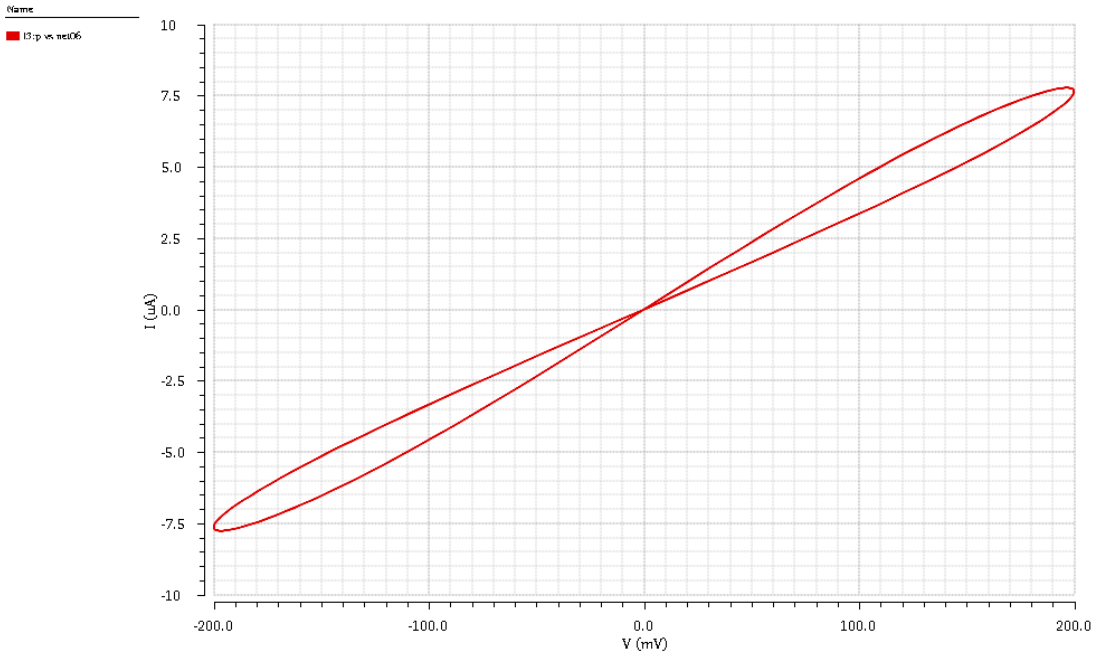

Figure 4.4 - Joglekar Hysteresis Loop with f=1Hz

In Figure 4.5, the hysteresis loop for a working frequency of $2Hz$ is represented.

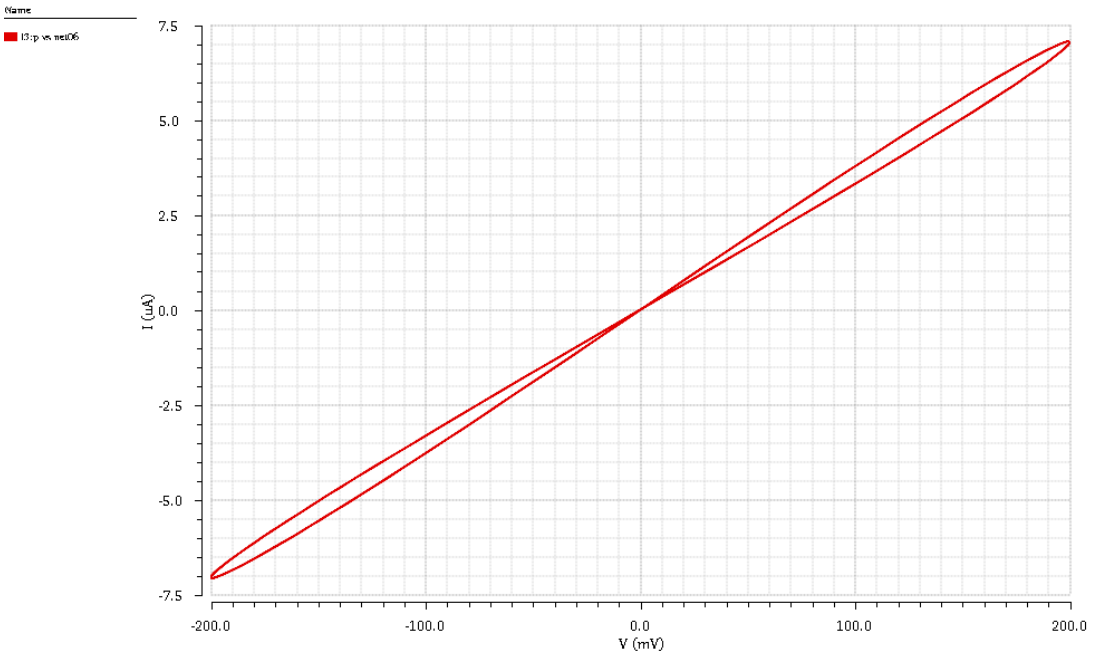

Figure 4.5 - Joglekar Hysteresis Loop with f=2Hz

B: Biolek Window Function

For the particular case of Biolek window function, the *window* variable is set to 2. In the first case, a frequency of $1Hz$, and a current of $100\mu A$ of amplitude were considered. In Figure 4.6 the transient response for the current, voltage and x_position are illustrated, whereas in Figure 4.7 the hysteresis loop is represented.

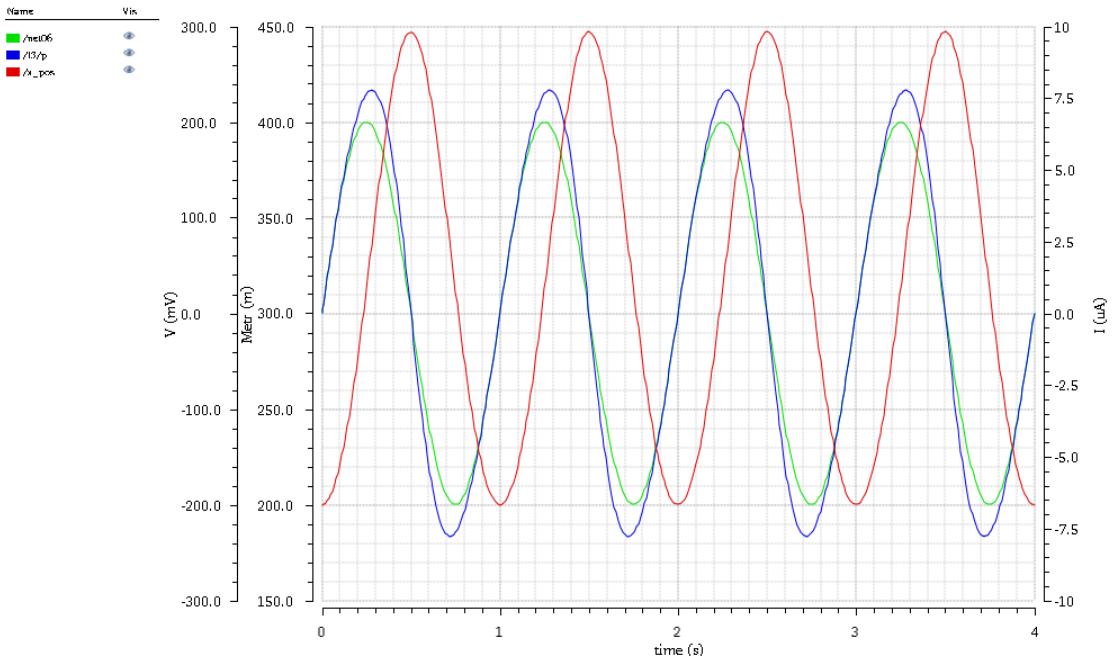

Figure 4.6 - Transient simulation of current, voltage and x_position with f=1Hz

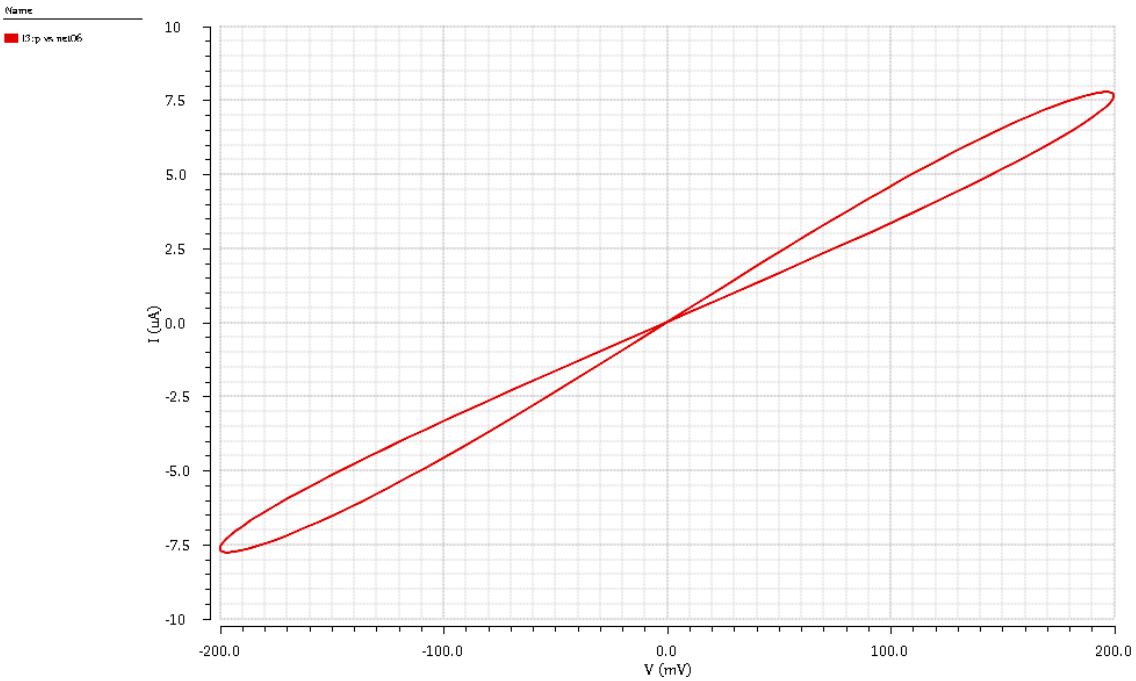

Figure 4.7 - Biolek Hysteresis Loop with f=1Hz

In Figure 4.8, the hysteresis loop for a working frequency of $2Hz$ is represented.

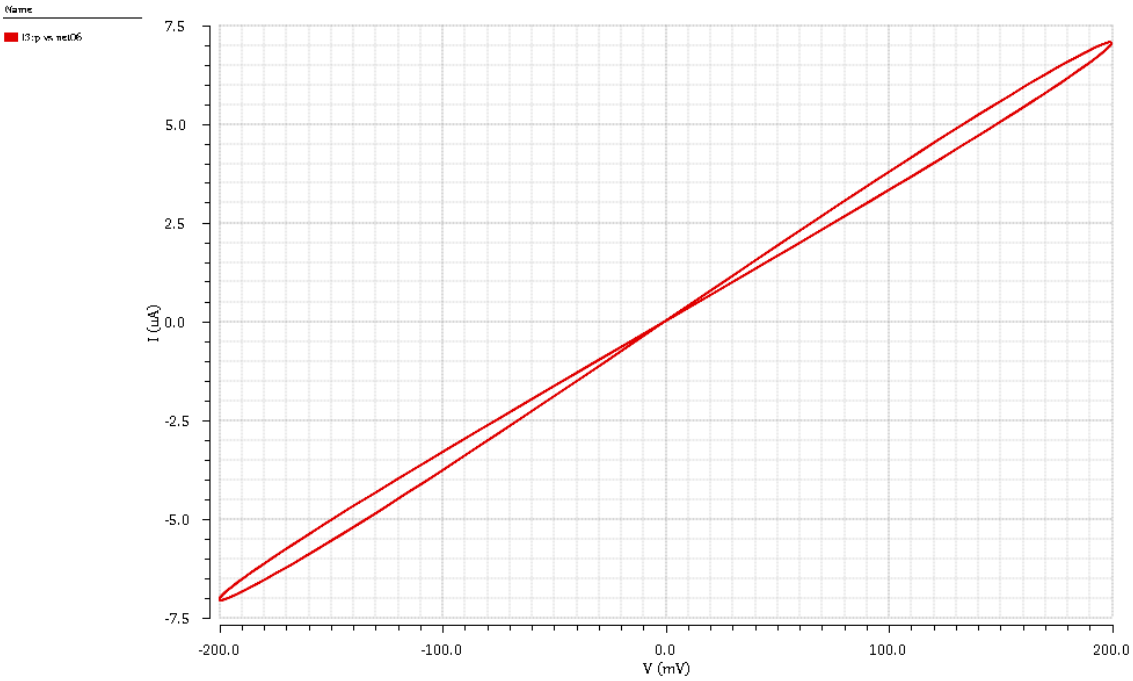

Figure 4.8 - Biolek Hysteresis Loop with f=2Hz

C: Prodromakis Window Function

For the particular case of Prodromakis window function, the *window* variable is set to 3. In the first case, a frequency of $1Hz$, and a current of $100\mu A$ of amplitude were considered. In Figure 4.9 the transient response for the current, voltage and x_position are illustrated, whereas in Figure 4.10 the hysteresis loop is represented.

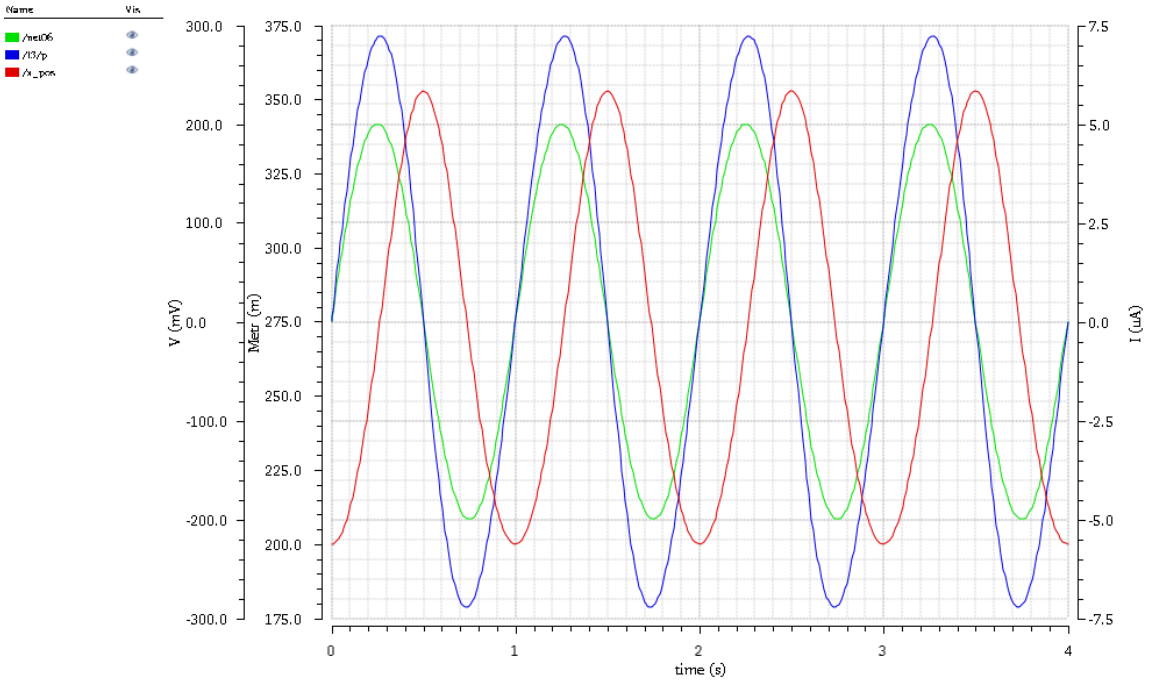

Figure 4.9 - Transient simulation of current, voltage and x_position with f=1Hz

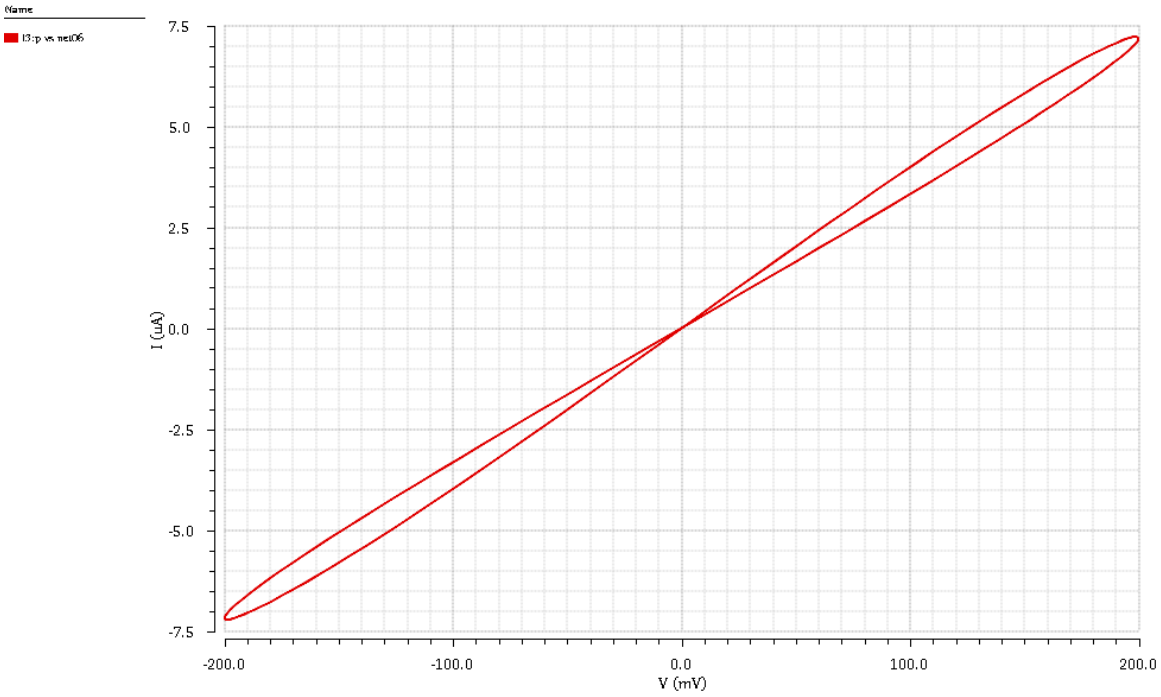

Figure 4.10 - Prodromakis Hysteresis Loop with f=1Hz

In Figure 4.11 the hysteresis loop for a working frequency of $2Hz$ is represented.

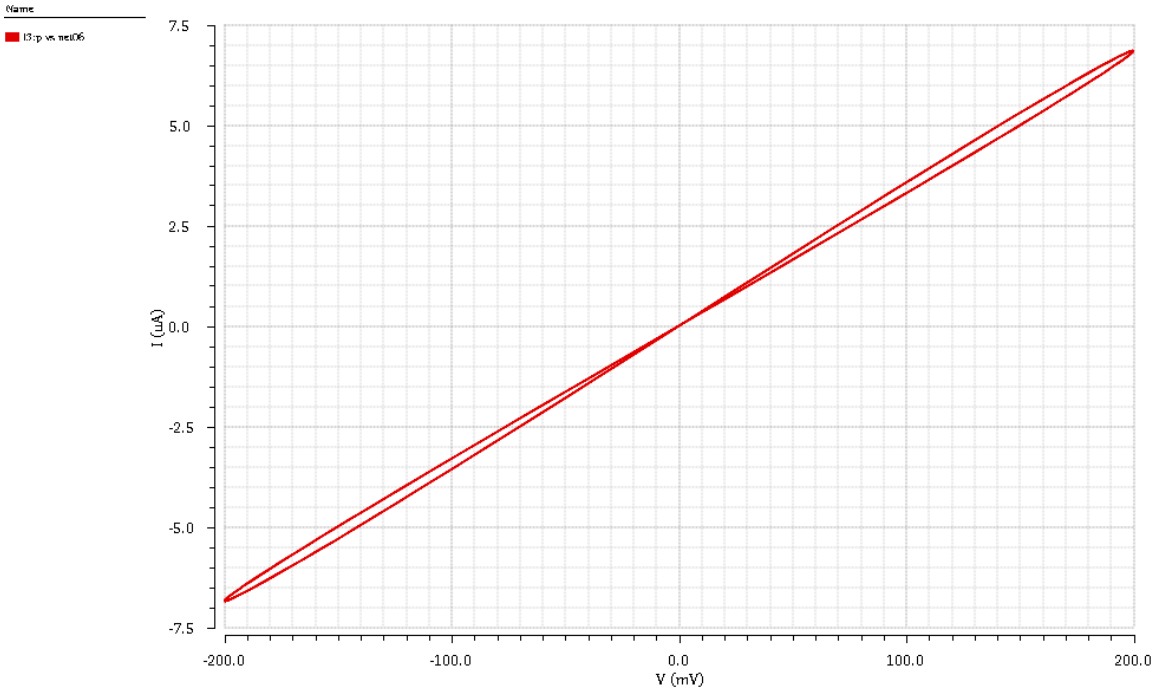

Figure 4.11 - Prodromakis Hysteresis Loop with f=2Hz

4.3 Memristor-Based Reactance-less Oscillators

Oscillators are electronic circuits that produce an output signal without any need of external source. They can convert direct current from any applied dc voltage to an alternating current signal and can be used as communications systems and digital systems for example [19].

A simple amplifier with positive feedback could represent an oscillator. The oscillators can be classified as sinusoidal and non-sinusoidal due to the waveform presented at the output.

An ideal sinusoidal oscillator should produce a sine-wave output signal with constant amplitude and no variation in frequency. Such oscillators can achieve frequency ranges from Hz to GHz. A non-sinusoidal oscillator, also known as relaxation oscillator, generates square or tri-angular waveforms at the output with frequency ranges from Hz to MHz.

Both types of oscillators depend on reactive elements to realize the oscillator function. Introducing the ability of the memristors to store the resistance value, increasing or decreasing the memristor resistance, eliminates the need to have reactive elements for charging and discharging. The inherent delay in the memristor response is exploited to realize the oscillator function [20].

4.3.1 Oscillator Structure

The architecture of the proposed oscillator is composed by two basic elements forming a voltage divider and a transfer function [21], as shown in Figure 4.12. The voltage across the element E_2 is given by eq. (33).

$$V_i(t) = V_o(t)\frac{R(E_2)}{R(E_1) + R(E_2)} \tag{33}$$

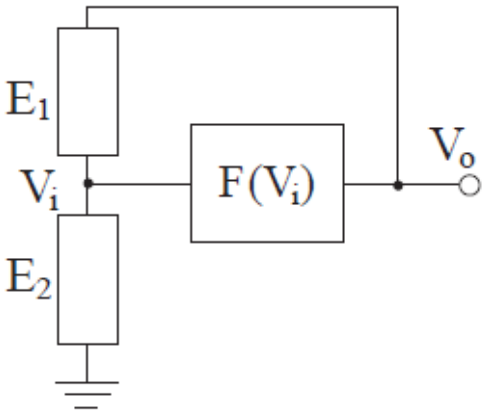

Figure 4.12 - General architecture adapted from [21]

In this case, the type of oscillator chosen, consists of having a resistance as element E_1 and a memristor as element E_2, that is, a floating resistor and a grounded memristor. Therefore, the voltage across the memristor is given by eq. (34). Assuming a positive configuration, the memristor is connected so that the magnitude of V_i increases when V_o is positive and decreases when V_o is negative. The threshold voltages V_p and V_n refers to a maximum and a minimum applied voltage, respectively. This transfer function is shown in Figure 4.13.

$$V_i(t) = V_o(t)\frac{R_m}{R_a + R_m} \tag{34}$$

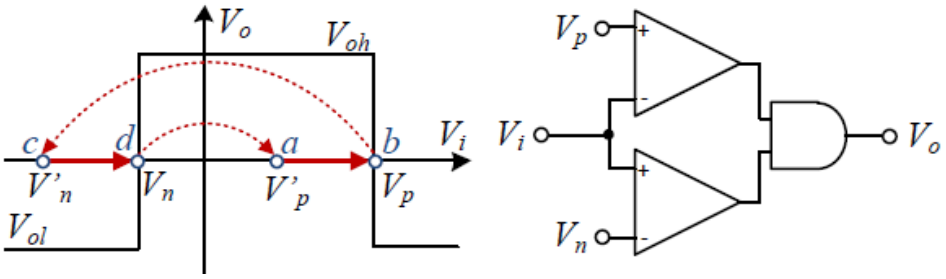

Figure 4.13 - Transfer function of F(V_i) with positive configuration and transitions between different operating points adapted from [21]

The operation principle of the oscillation is based as follows: At point a, the output voltage V_o is positive and equal to V_{oh}. The resistances R_a and R_m will increase but R_m increases in a faster way.

At point b, the input voltage V_i is equal to V_p and the value of V_o switch from V_{oh} to V_{ol}. Therefore, the operation point will jump to point c.

At this point, the values of the resistances of the two elements will decrease because V_o is negative. The value of R_b will decrease in a faster way since it is an element connected to the ground. The input voltage will decrease, and the operating point will move to d.

At last, at point d, the value of V_i just cross V_n and the voltage V_o switch its value from V_{ol} to V_{oh}. The operating point will move to point a and the oscillation will continue in this cycle.

So, the proposed schematic for the oscillator is illustrated in Figure 4.14.

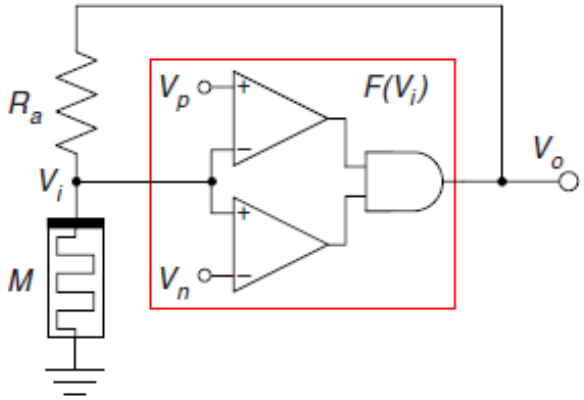

Figure 4.14 - Proposed memristor-based reactance-less oscillator adapted from [20]

From eq. (33), for an input voltage $V_i = V_p$ and an output voltage $V_o = V_{oh}$, the transition resistance is defined by eq. (34).

$$R_{mp} = R_a \frac{V_p}{V_{oh} - V_p} \tag{34}$$

Similarly, for an input voltage $V_i = V_n$ and an output voltage $V_o = V_{ol}$, the transition resistance is defined by eq. (35).

$$R_{mn} = R_a \frac{V_n}{V_{ol} - V_n} \tag{35}$$

The resistances R_{mp} and R_{mn} should be selected in order to verify the condition $R_{ON} < R_{mn} < R_{mp} < R_{OFF}$. Based on the operation principle described before, the oscillation will occur if V_p and V_n are selected in a way to verify eq. (36).

$$V_p - V_n \frac{V_{oh}}{V_{ol}} > 0 \tag{36}$$

The time required for the memristor to change its resistance from R_{mn} to R_{mp} determines the time required by the circuit to change its state from *a* to *b* [20]. Assuming the mathematical model of the HP memristor defined by eq. (37), the derivative in order of time is represented by eq. (38).

$$R_m^2(t) = R_o^2 \pm 2k' \int_0^t V_m(\tau) d\tau \tag{37}$$

$$R_m dR_m = k' V_i(t) dt \tag{38}$$

Substituting eq. (34) into eq. (38) and integrating, the equation obtained is defined by eq. (39).

$$\int_0^{T_H} dt = \frac{1}{k' V_{oh}} \int_{R_{mn}}^{R_{mp}} (R_m + R) dR_m \tag{39}$$

Solving the integration, the time of the positive half cycle is obtained by eq. (40).

$$T_H = \frac{R_{mp}^2 - R_{mn}^2 + 2R\left(R_{mp} + R_{mn}\right)}{2k' V_{oh}} \tag{40}$$

The time of the negative half cycle is obtained by eq. (41).

$$T_L = \frac{R_{mn}^2 - R_{mp}^2 + 2R\left(R_{mn} - R_{mp}\right)}{2k' V_{ol}} \tag{41}$$

Substituting eq. (34) and eq. (35) into eq. (40) and eq. (41), the duty cycle expression is represented by eq. (42).

$$D = \frac{|V_{ol}|}{V_{oh} - V_{ol}} \tag{42}$$

The same substitutions can derive the frequency expression of the oscillator given by eq. (43).

$$f_0 = \frac{2Dk'V_{oh}\left(V_{oh} - V_p\right)^2 (V_n - V_{ol})^2}{R_a^2\left(V_pV_{ol} + V_nV_{oh}\right)\left(2V_{oh}V_{ol} - V_{ol}V_p - V_{oh}V_n\right)} \tag{43}$$

4.3.2 VerilogA Models for the oscillator elements

In this section, the implementation in VerilogA of the models for the operational amplifier and the digital gates is presented. The implementation of the operational amplifier, presented in listing 4, was considered. The parameters used were based on the typical characteristics of the amplifier, which means that the value of the output resistance is low, the values of the input resistance and the gain are high. The value for the input resistance was 10Ω, the value of the output resistance was $1M\Omega$ and the gain was 10^5.

```
 `include “constants.vams”
 `include “disciplines.vams”

module opamp (inp, inn, out, vsupply_p, vsupply_n, vref)
input inp, inn, vsupply_p, vsupply_n, vref;
output out;
electrical inp, inn, vsupply_p, vsupply_n, vref;
parameter real gain=1e5;
parameter real Rin=1M;
parameter real Rout=10;
real Vdc, Vamp;
analog begin
    Vamp=(V(vsupply_p)-V(vsupply_n))/2;
    Vdc=(V(vsupply_p)+V(supply_n))/2;
    V(inp, inn)<+Rin*I(inp, inn);
    V(out)<+Vamp*tanh(gain*V(inp,inn)/Vamp)+Vdc;
    V(out)<+Rout*I(out);
end
endmodule
```

Listing 4: VerilogA code for the implementation of opamp

In Figure 4.15, the schematic for the transient analysis is presented. The testing circuit was based on a non-inverting configuration. The input voltage used was $100mV$. In the negative feedback loop, the values of the resistances used were $1k\Omega$ and $10k\Omega$.

These values were chosen just to simulate the behavior of this circuit. The operational amplifier was supplied with a positive input voltage of $1.2V$ and a negative input voltage of $-1.2V$.

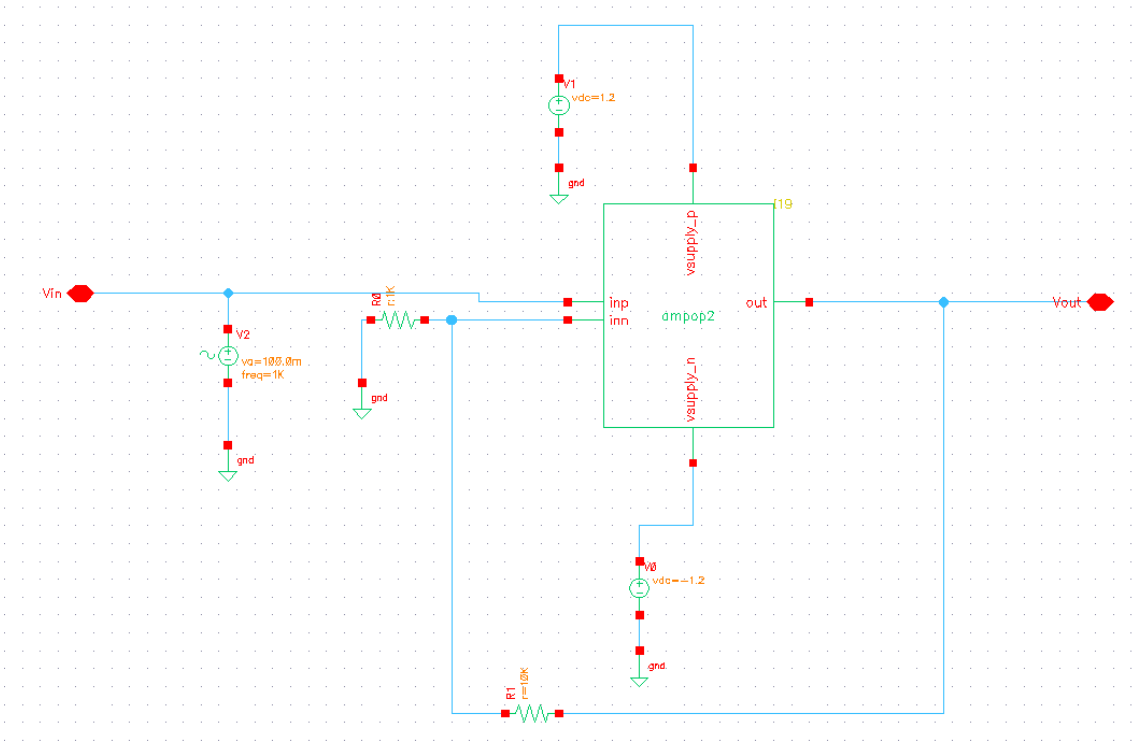

Figure 4.15 - Schematic of the circuit for transient simulation

The transient response is presented in Figure 4.16. The value of the output voltage is approximately $1.1V$, which means that this amplifier presents a gain of 11.

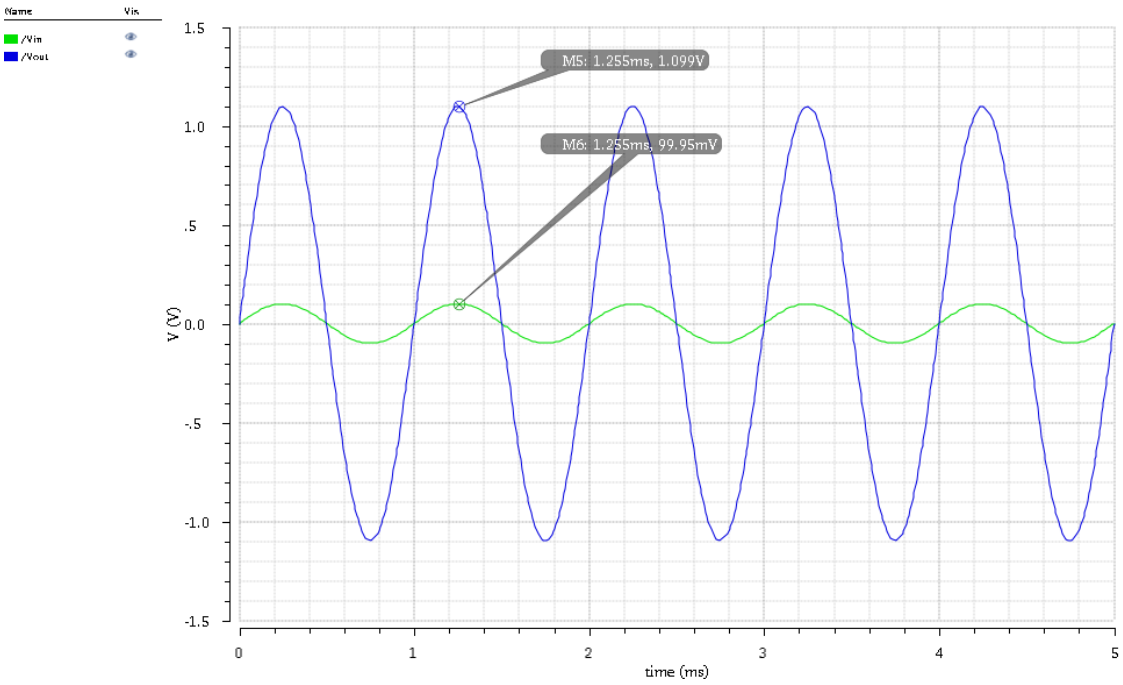

Figure 4.16 - Transient response of Vin and Vout

After simulating the behavior of the operational amplifier, it is necessary to implement a comparator circuit. In this case, the comparator circuit used, is based on two operational amplifiers and output control logic. The output control logic was based on an *and* gate circuit.

To obtain an *and* gate, a *nand* gate with an inverter at the output is presented. The schematic of this circuit is illustrated in Figure 4.17.

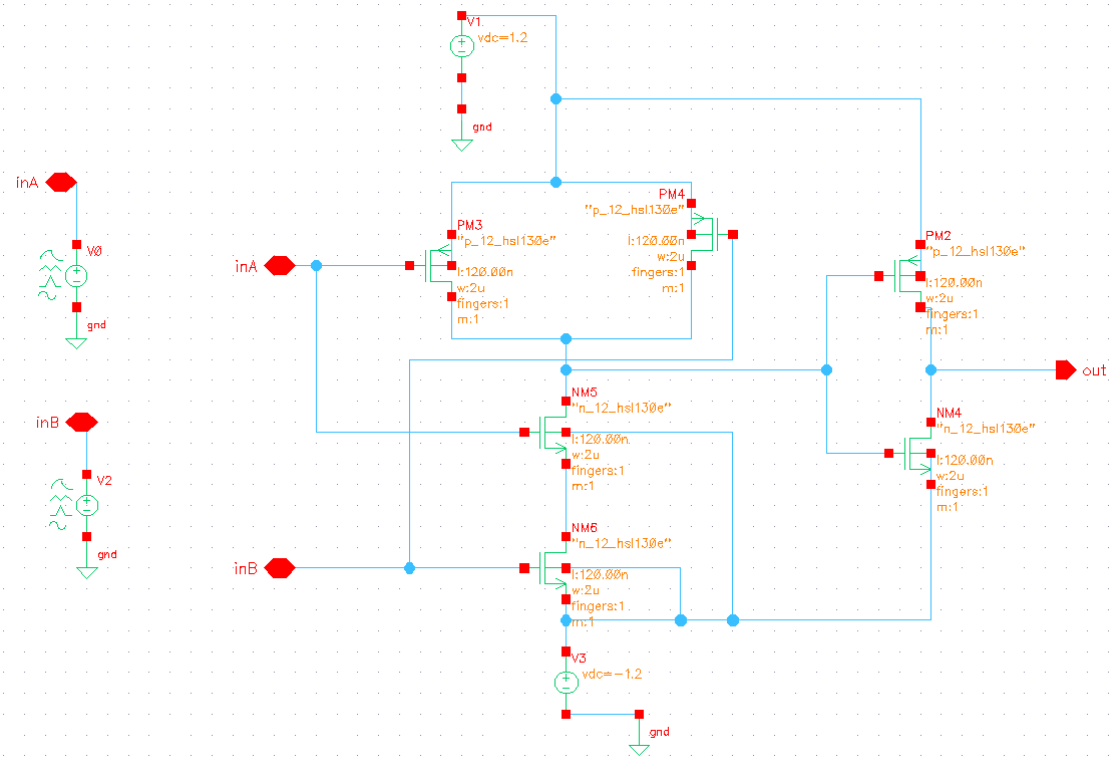

Figure 4.17 - Schematic of the and gate for transient simulation

The simulation for the transient behavior is presented in Figure 4.18. The green signal is the output of the *nand* gate, which means that the output is 0 when the both inputs are 1, otherwise the output is always 1.

The blue signal is the output of the *and* gate, which means that its behavior is the opposite of the *nand* gate. So, the output is always 1 when the both inputs are 1, otherwise the output is always 0.

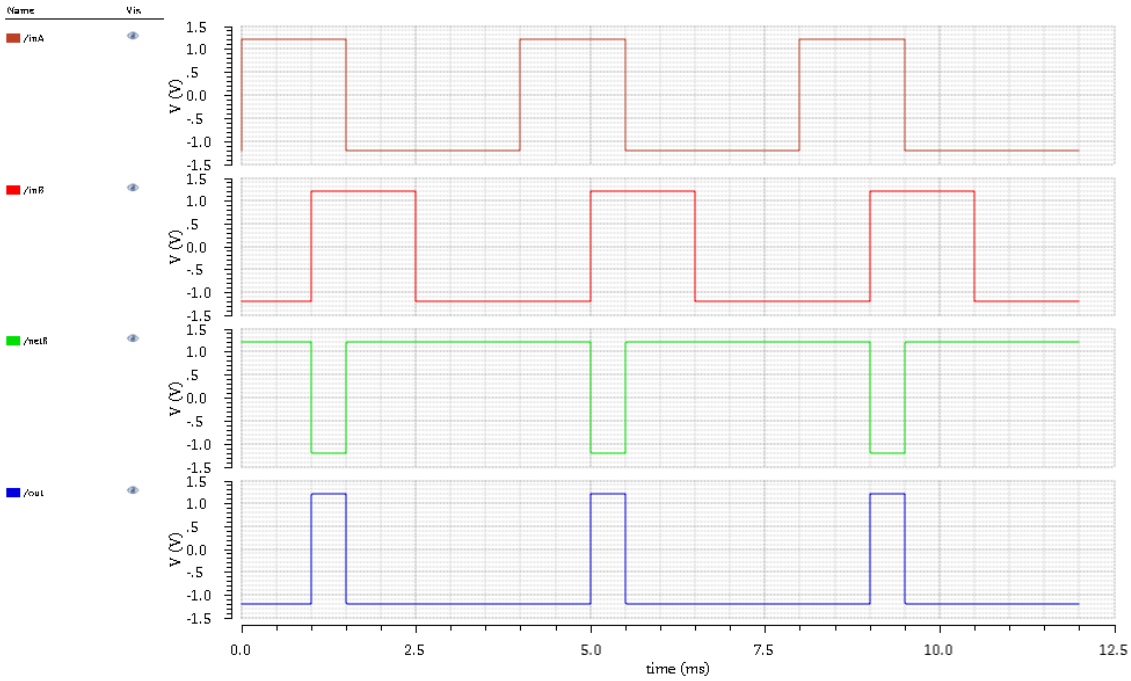

Figure 4.18 - Transient response of the and gate

The schematic of the comparator is presented in Figure 4.19. The operational amplifiers were supplied by $1.2V$. The input voltage source V_{in} was set to be a pulse wave. The reference voltages V_p and V_n were set to be $0.7V$ and $0.3V$, respectively.

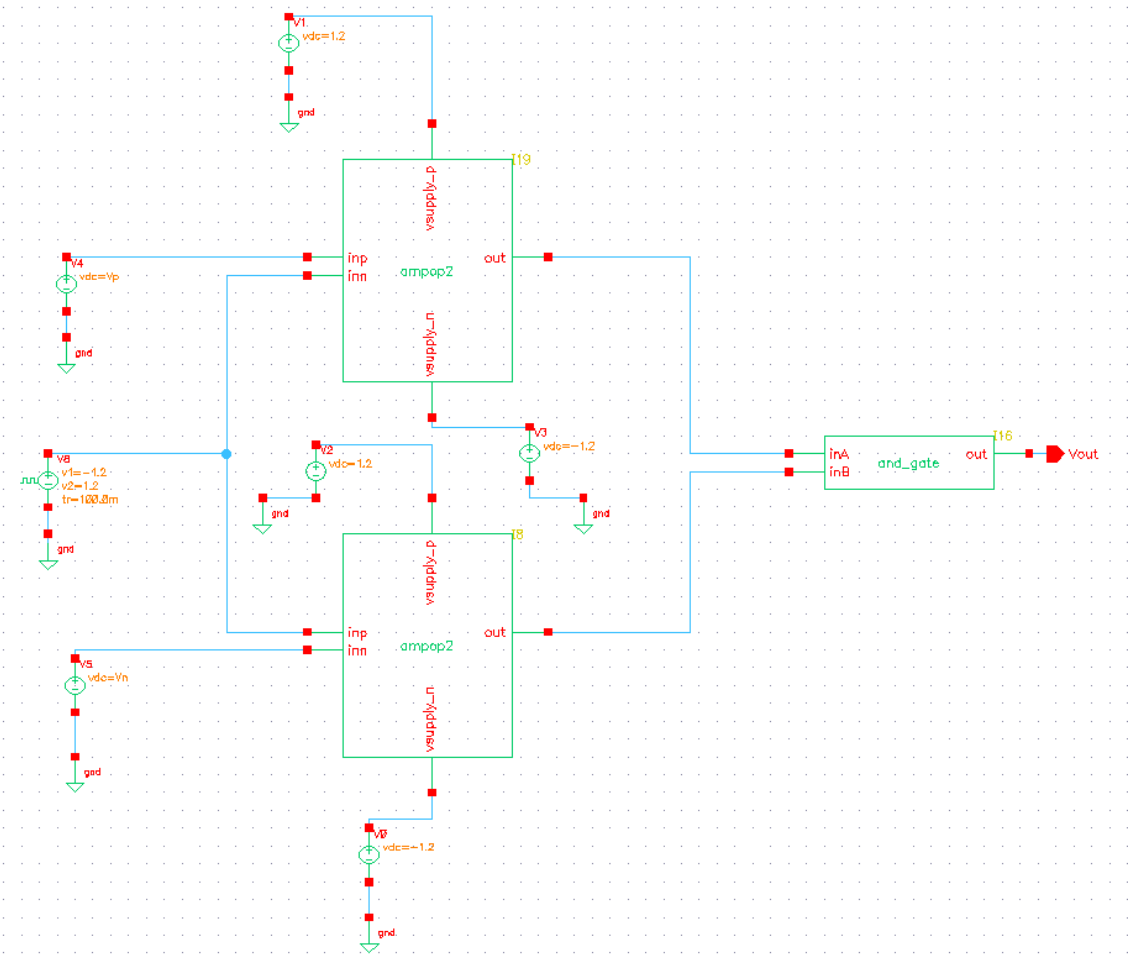

Figure 4.19 - Schematic of the comparator circuit

The simulation response of the comparator is illustrated in Figure 4.20. The green signal represents the input voltage of the comparator, which is defined by a pulse source, between $-1.2V$ and $1.2V$. The red and yellow signals represent the reference voltages, with $0.7V$ and $0.3V$, respectively. The operation principle is determined by the comparation between the input voltage and the reference voltages.

If the input voltage is between the reference voltages, the output voltage represented by the blue signal is always 1, in this case achieves the value of $1.2V$. Otherwise, the value of the output is always 0, which in this case represents the value of $-1.2V$.

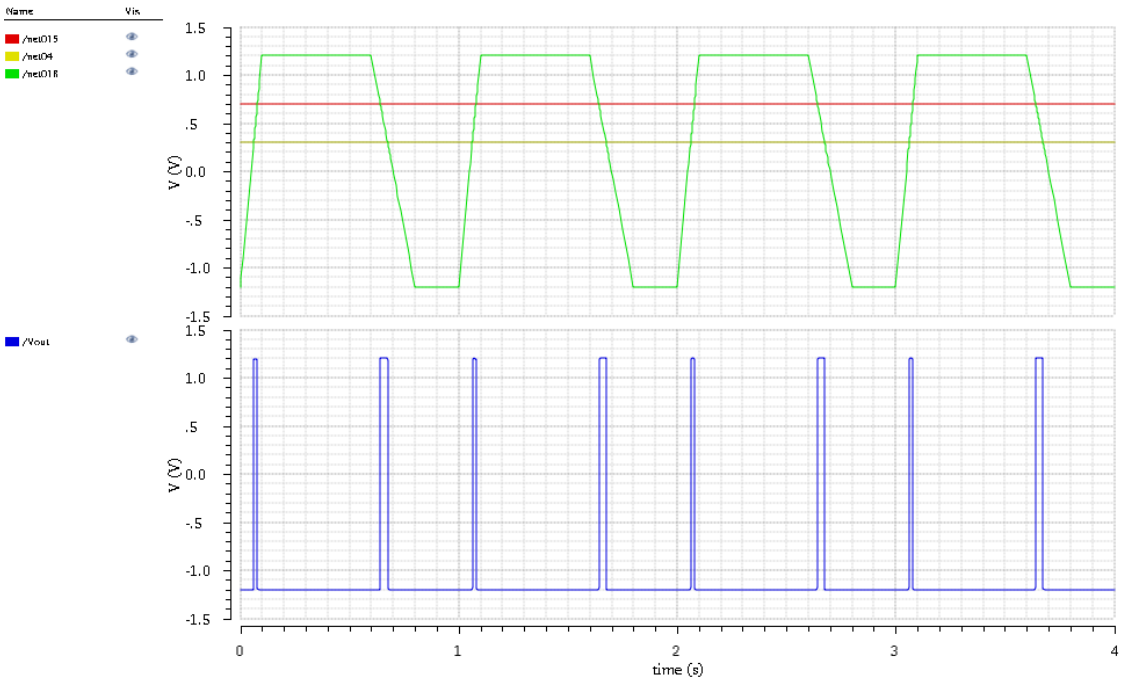

Figure 4.20 - Transient response of the comparator

In Figure 4.21, the schematic of the memristor circuit with the comparator is presented. The simulation was made considering the open loop configuration, and the results are presented in Figure 4.22.

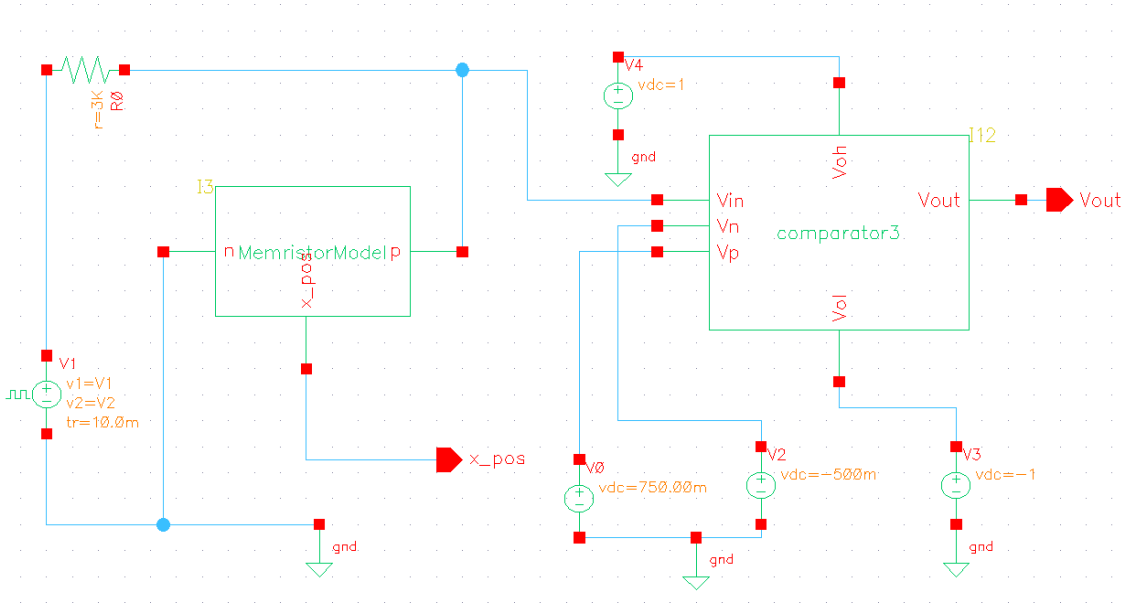

Figure 4.21 - Schematic of the circuit for open loop simulation

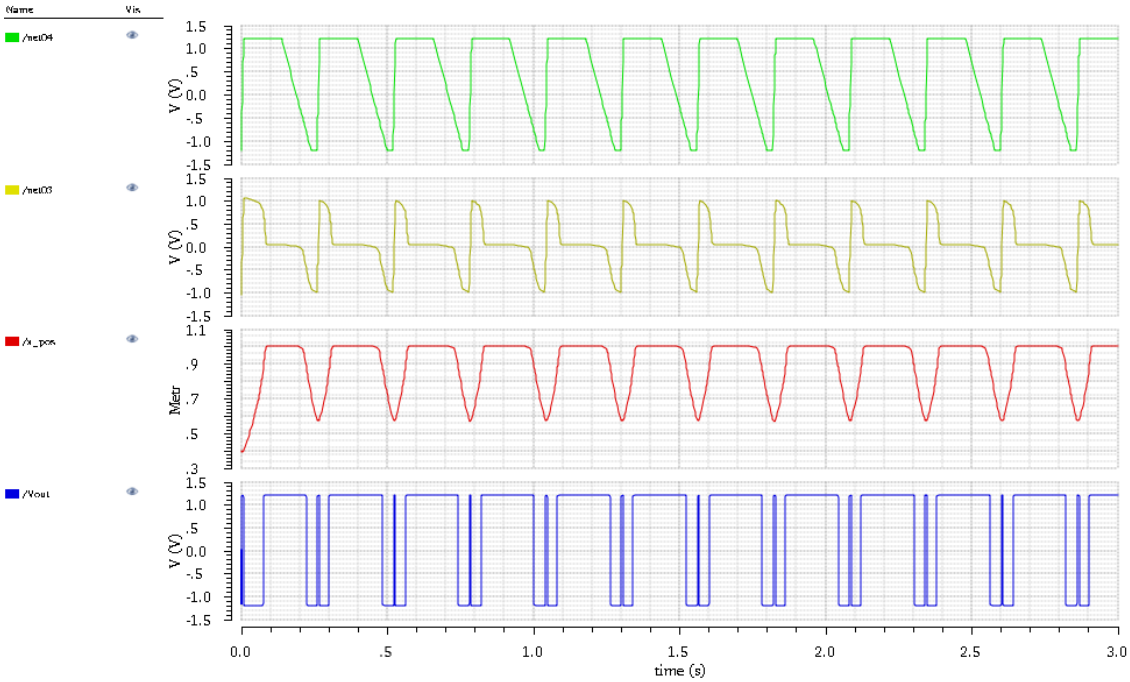

Figure 4.22 - Transient simulation of the open loop circuit

4.3.3 Conclusion

In this section, the results for the implemented memristor models were presented using the VerilogA language. The simulation for the different models confirm the behavior previously presented. The introduction of these models was used to implement an oscillator without using reactive elements.

The different blocks that constitute the oscillator were properly implemented showing the correct behavior of each one. The final simulation of the oscillator was not possible due to convergence problems.

5. Conclusion and future work

In chapter 2, the state of the art was presented, with the description of the several linear and non-linear memristor models. The analytical characterization was considered based on the literature.

In chapter 3, the implementation of the presented models in Matlab was the focus. The results obtained consider the hysteresis loops where the relation between current and voltage is achieved. The pinched hysteresis loop depends on the values of the resistances R_{ON} and R_{OFF}. As the frequency decreases, the length of the memristor is more reached and the hysteresis loop become larger. The numerical errors presented in Euler method were improved with the introduction of the Runge-Kutta method.

In chapter 4, the models for elements of an oscillator using a memristor were developed in VerilogA. The simulation results of these models agree with the expected ones. The open loop simulation of the memristor circuit with the comparator was made. This configuration would be the base of the oscillator circuit. Considering a pulse source in the input, the results at the output agree with the expected. The closed loop simulation presented problems of convergence that were not possible to solve and may be considered as candidate for the future work.

Further future work to be considered is the implementation in VerilogA of more accurate memristor models, e.g., TEAM and the polynomial model.

6. References

[1] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, “The missing memristor found,” vol. 453, no. May, pp. 80–83, 2008.

[2] W. Hang and P. Yun-, “A M odel Checking Tool Based on D iscrete T im ed Automata,” *2017 Int. Conf. Energy, Commun. Data Anal. Soft Comput.*, vol. 9348, no. 2, pp. 80–83, 2008.

[3] G. S. Rose, “Overview: Memristive devices, circuits and systems,” *Circuits Syst. (ISCAS), Proc. 2010 IEEE Int. Symp.*, pp. 1955–1958, 2010.

[4] A. F. Adzmi and S. H. Herman, “Memristor Spice Model for Designing Analog Circuit,” pp. 78–83, 2012.

[5] K. Agashe, N. Sarwade, S. Joshi, P. Soman, and T. Engineering, “Comprehensive study of current controlled Memristor models,” vol. 7, no. 2, pp. 1376–1381, 2016.

[6] J. Albo-canals and G. E. Pazienza, “How to Teach Memristors in EE Undergraduate Courses,” pp. 345–348, 2011.

[7] Y. N. Joglekar and S. J. Wolf, “The elusive memristor : properties of basic electrical circuits The elusive memristor : properties of,” 2009.

[8] V. M. Mladenov and S. M. Kirilov, “Memristor modeling in matlab ® & pspice ®,” vol. 8, no. Cd.

[9] V. Keshmiri, “Institutionen för systemteknik Department of Electrical Engineering A Study of the Memristor Models and Applications,” 2014.

[10] R. K. Singh, “Window Function Analysis of Nonlinear Behaviour of Fourth Fundamental Passive Circuit Element : Memristor,” vol. 12, no. 3, pp. 58–63, 2017.

[11] N. D. Kinetics, T. Prodromakis, and B. P. Peh, “A Versatile Memristor Model With,” vol. 58, no. 9, pp. 3099–3105, 2011.

[12] S. Kvatinsky, E. G. Friedman, A. Kolodny, S. Member, and U. C. Weiser, “TEAM : ThrEshold Adaptive Memristor Model,” vol. 60, no. 1, pp. 211–221, 2013.

[13] R. K. Sidhu, “Different Models of Memristor,” vol. 4, no. 06, pp. 991–994, 2015.

[14] A. Ascoli, F. Corinto, V. Senger, and R. Tetzlaff, “Memristor Model Comparison,” pp. 89–105, 2013.

[15] M. D. R. Rkf, “Método de Runge-Kutta-Fehlberg (RKF45),” p. 75240.

[16] J. H. Mathews and K. D. Fink, “mcgraw-hill-numerical-methods-using-matlab.pdf.” 1999.

[17] S. Kvatinsky, K. Talisveyberg, D. Fliter, A. Kolodny, U. C. Weiser, and E. G. Friedman, “Models of memristors for SPICE simulations,” *2012 IEEE 27th Conv. Electr. Electron. Eng. Isr. IEEEI 2012*, pp. 1–5, 2012.

[18] T. Wang and J. Roychowdhury, “Well-Posed Models of Memristive Devices,” 2016.

[19] G. B. A. E. C. B. Dbedeb, “A Low Frequency Oscillator Structure,” pp. 978–980, 2009.

[20] M. Affan Zidan, H. Omran, A. G. Radwan, and K. N. Salama, “Memristor-based reactance-less oscillator,” *Electron. Lett.*, vol. 47, no. 22, p. 1220, 2011.

[21] “Memristor Circuits and Systems Thesis by Mohammed Affan Zidan In Partial Fulfillment of the Requirements For the Degree of Doctor of Philosophy,” 2015.

Appendix A

LTspice Macromodels

```
**********************************************
* HP Memristor SPICE Model
* For Transient Analysis only
* created by Zdenek and Dalibor Biolek
**********************************************
* Ron, Roff - Resistance in ON / OFF States
*
* Rinit - Resistance at T=0
*
* D - Width of the thin film
*
* uv - Migration coefficient
*
* p - Parameter of the WINDOW-function for
*     modeling nonlinear boundary conditions
*
* x - W/D Ratio, W is the actual width
*     of the doped area (from 0 to D)
*
**********************************************
.SUBCKT memristor plus minus PARAMS:
+ Ron=100 Roff=16K Rinit=11K D=10N uv=10F p=10

**********************************************
* DIFFERENTIAL EQUATION MODELING *
**********************************************
Gx 0 x value={I(Emem)*uv*Ron/D**2*f(V(x),p)}
Cx x 0 1 IC={(Roff-Rinit)/(Roff-Ron)}
Raux x 0 1000000

**********************************************
* RESISTIVE PORT OF THE MEMRISTOR *
**********************************************
Emem plus aux value={-I(Emem)*V(x)*(Roff-Ron)}
Roff aux minus {Roff}

**********************************************
* FLUX COMPUTATION *
**********************************************
Eflux flux 0 value={SDT(V(plus,minus))}

**********************************************
* CHARGE COMPUTATION *
**********************************************
Echarge charge 0 value={SDT(I(Emem))}

**********************************************
* WINDOW FUNCTIONS
```

```
* FOR NONLINEAR DRIFT MODELING *
**********************************************
* window function, according to Joglekar
.func f(x,p)={1-(2*x-1)**(2*p)}

* window function, according to Biolek
.func f(x,i,p)={1-(x-stp(-i))**(2*p)}

* window function, according to Prodromakis
.func f(x,p)={1-(((x-0.5)**2)+0.75)**p}

* window function, according to Strukov
.func f(x,p)={x-x*2}

.ENDS memristor
```

Appendix B

```
%Linear Memristor models
%Window functions applied
clear all;
close all;
p=10;
v0=1.2;
t=0:1e-3:1;
omega=2*pi;
v=v0*sin(omega*t);
D=10e-9;
j=1;
Miuv=10e-15;
Ron=0.1e3;
Roff=16e3;
Rin=11e3;
w(1)=0.3*D; %((Roff-Rin)/(Roff-Ron))*D;
x(1)=w(1)/D;
m(1)=Ron*(x(1))+Roff*(1-x(1));
fw(1)=x-x.^2;                          %Strukov window
fw(1)= 1-(2*x(1)-1)^(2*p);             %Joglekar window
fw(1)= j*(1-((x(1)-0.5)^2+0.75)^p);    %Prodromakis window
fw(1)= 1-((x-1)^(2*p));                %Biolek window
fw(1)= 1-((x(1))^(2*p));
i(1)=v(1)/m(1);

for index=2:length(t)
    v_d(index)= (Miuv*Ron*i(index-1)*fw(index-1))/D;
    w(index)= v_d(index)*(t(index)-t(index-1))+w(index-1);
    x(index)= w(index)/D;
    fw(index)=x(index)-x(index).^2;                %Strukov window
    fw(index)= 1-(2*x(index)-1)^(2*p);             %Joglekar window
    fw(index)= j*(1-((x(index)-0.5)^2+0.75)^p);    %Prodromakis window
    if i(index-1)<0                                %Biolek
        fw(index)=1-(x(index)-1)^(2*p);
     else
        fw(index)=1-(x(index))^(2*p);
     end
    m(index)=Ron*(w(index)/D)+Roff*(1-w(index)/D);
```

```
    i(index)=v(index)./m(index);

end

figure(1)
plot(v,i)
hold on
grid on

A=dlmread('joglekar_p10.txt','\t',[1,0,155,2]);
v=A(:,2);
i=A(:,3);
plot(v,i);
xlabel({'Voltage','(V)'});
ylabel({'Current','(uA)'});
title('Joglekar I-V Hystereris Loop');
grid on
hold on

A=dlmread('biolek_p10.txt','\t',[1,0,141,2]);
v=A(:,2);
i=A(:,3);
plot(v,i);
xlabel({'Voltage','(V)'});
ylabel({'Current','(uA)'});
title('Biolek I-V Hystereris Loop');
grid on
hold on

A=dlmread('prodromakis_p10.txt','\t',[1,0,127,2]);
v=A(:,2);
i=A(:,3);
plot(v,i);
xlabel({'Voltage','(V)'});
ylabel({'Current','(uA)'});
title('Prodromakis I-V Hystereris Loop');
grid on
hold on

A=dlmread('strukov_p10.txt','\t',[1,0,110,2]);
v=A(:,2);
i=A(:,3);
plot(v,i);
xlabel({'Voltage','(V)'});
ylabel({'Current','(uA)'});
title('Strukov I-V Hystereris Loop');
grid on
```

Appendix C

```
%Team model
function []=team_model2()

close all;

%Parameters
ncycles=1;
```

```
A=0.008;
f=1e3;
xinit=0.5; % the initial state condition [0:1]
Roff=1e3;
Ron=50;
fi=0;

%Team parameters
aon=1.8;
aoff=1.2;
alphaon=3;
alphaoff=3;
kon=-4.68e-4;  %negative
koff=1.46;   %positive
ion=-115e-6;   %negative
ioff=890e-6;   %positive
xon=0.2;
xoff=2.4;
wc=107e-3;
k=1;
points=2e2;
tvec=[0 ncycles/f];
t=linspace(tvec(1),tvec(2),points);
i=A*sin(2*pi*f*t+fi);
deltat=t(2)-t(1);
x(1)=xinit;
R=getR(x(1),Ron,Roff,xon,xoff,k);
v(1)=R*i(1);
dx(1)=0;

for j=2:(length(t))
    dx(j)=gdxdt(i(j-1),koff,kon,ioff,ion,x(j-1),wc,aoff,aon)*deltat;
    x(j)=x(j-1)+dx(j);
    R=getR(x(j),Ron,Roff,xon,xoff,k);
    v(j)=i(j)*R;

end

figure(1);
plot(v,i);
sx=strcat('IV Hysteresis Loop for frequency of {}', num2str(f),' Hz');
title(sx);
xlabel('v(V)');
ylabel('i(A)');
grid on

figure(2);
plot(i,dx);
title('dx vs time');
ylabel('dx');
xlabel('t (s)');
grid on

end

function dxdt=gdxdt(i,koff,kon,ioff,ion,x,wc,aoff,aon)
if (i>ioff)
    dxdt=(koff*(i/ioff-1)^3).*exp(-exp((x-aoff)/wc));
elseif (i<ion)
    dxdt=(kon*(i/ion-1)^3).*exp(-exp((x-aon)/wc));
```

```
else
    dxdt=0;
end
end

function R=getR(x,Ron,Roff,xon,xoff,k)
if (k==1)
    R=Ron+(Roff-Ron)*(x-xon)/(xoff-xon);  %linear
else
    lambda=reallog(Roff/Ron);
    R=Ron*exp(lambda*(x-xon)/(xoff-xon));  %exponential
end
end
```

Appendix D

```
%Runge-kutta 4.5 Method
function [i] = hf_rungekutta45(freq,ncycle,h,nfig)
format long
k=2;
p=5;
j=1;
epsilon=1e-17;
w=5e-9;
i=1;
t(1)=0;
omega=2*pi*freq;
Miuv=10e-15;
D=10e-9;
Ron=0.1e3;
Roff=16e3;

fi=0; % voltage phase

x=w/D;
Rini=Ron*(x)+Roff*(1-x);
iy=cos(fi)/Rini;
t_final=ncycle/freq

while t(i)<t_final

    k1=h*f(t(i),w(i),p,j);
    k2=h*f(t(i)+h/4,w(i)+k1/4,p,j);
    k3=h*f(t(i)+3*h/8,w(i)+3*k1/32+9*k2/32,p,j);
    k4=h*f(t(i)+12*h/13,w(i)+1932*k1/2197-
7200*k2/2197+7296*k3/2197,p,j);
    k5=h*f(t(i)+h,w(i)+439*k1/216-8*k2+3680*k3/513-845*k4/4104,p,j);
    k6=h*f(t(i)+h/2,w(i)-8*k1/27+2*k2-3544*k3/2565+1859*k4/4104-
11*k5/40,p,j);
    w1=w(i)+25*k1/216+1408*k3/2565+2197*k4/4104-k5/5;
    w2=w(i)+16*k1/135+6656*k3/12825+28561*k4/56430-9*k5/50+2*k6/55;
    R=abs(w1-w2)/h ;
    if R==0
        delta=1;
    else
        delta=0.84*(epsilon/R)^(1/4);
    end
```

```
     if R<=epsilon
        i=i+1;
        w(i)=w1;
        MR=Ron*(w(i)/D)+Roff*(1-w(i)/D);
        h=delta*h;
        t(i)=t(i-1)+h;
        ix(i)=sin(omega*t(i))/MR;
        iy=ix(i);
     else
         h=delta*h;

     end
end

vx=sin(omega*t);
figure(nfig)
plot(vx,ix);
hold on
grid on

function dw=f(t,w,p,j)
    x=w/D;
    MR=Ron*(w/D)+Roff*(1-w/D);
    iz=sin(omega*t)/MR;
    dw=Miuv*Ron/D*iz;
    if k==1
        dw=dw*(1-(2*x-1)^(2*p));             %joglekar
    elseif k==2
        dw=dw*j*(1-((x-0.5)^2+0.75)^p);    %prodromakis
    elseif k==3
        if iz<0
            dw=dw*(1-((x-1.0)^(2*p)));       %biolek i<0
        else
            dw=dw*(1-x^(2*p));              %biolek i>0

        end
    end
end

end
```

Appendix E

```
%Euler Method
function [i] = Euler(freq,ncycle,h,nfig)
format long
k=1;
p=5;
j=1;

w(1)=5e-9; % initial
i=1;
%freq=2;
omega=2*pi*freq;
Miuv=10e-15;
D=10e-9;
```

```
Ron=0.1e3;
Roff=16e3;
fi=0; % voltage phase
% fprintf('Step %d: t=%6.4f, w=%18.15f\n',i,t,q);
t_final=ncycle/freq
% h=t_final/npoints
t=0:h:t_final;
v=cos(omega*t+fi);
Rini=Ron*(w(1)/D)+Roff*(1-w(1)/D);
ix(1)=v(1)/Rini;
iy=ix(1)

while t(i)<t_final
    %h=min(h,t_final-t(i));
    dw=h*f(t(i),p,j,iy);
    i=i+1;
    w(i)=w(i-1)+dw;
    R=Ron*(w(i)/D)+Roff*(1-w(i)/D);
    ix(i-1)=v(i-1)/R;
    iy=ix(i-1);
        %fprintf('h=%f?\n',h)
end
figure(10);
plot(w)
vx=cos(omega*(t(2:end)));
figure(nfig)
plot(vx,ix,'r');
hold on
grid on

function dw=f(t,p,j,i)
    x=w/D;
    dw=Miuv*Ron/D*i;
    if k==1
        dw=dw*(1-(2*x(1)-1)^(2*p));            %joglekar
    elseif k==2
        dw=dw*j*(1-((x(1)-0.5).^2+0.75)^p);    %prodromakis
    elseif k==3
        if i<0
            dw=dw*(1-((x(1)-1).^(2*p)));       %biolek i<0
        else
            dw=dw*(1-x(1).^(2*p));             %biolek i>0

        end
    end
end

end
```

Appendix F

```
analog function real smoothabs;
    real x,e; input x,e;
    smoothabs=sqrt(x*x+e)-sqrt(e);
endfunction

analog function real dsmoothabs;
    real x,e; input x,e;
    dsmoothabs =x/sqrt(x*x+e);
endfunction

analog function real ddsmoothabs;
    real x,e; input x,e;
    dsmoothabs=(x/(pow(smoothabs(x,e),2)))*dsmoothabs(x,e)+(1/smoothabs(x,e));
endfunction

analog function real smoothclip;
    real x,e; input x,e;
    smoothclip=0.5*(smoothabs(x,e)+x);
endfunction

analog function real dsmoothclip;
    real x,e; input x,e;
    dsmoothclip=0.5*dsmoothabs(x,e)+0.5;
endfunction

analog function real ddsmoothclip;
    real x,e; input x,e;
    dsmoothclip=0.5*ddsmoothabs(x,e)+0.5;
endfunction

analog function real smoothstep;
    real x,e; input x,e;
    smoothstp=dsmoothclip(x,e);
endfunction

analog function real dsmoothstep;
    real x,e; input x,e;
    dsmoothstep=ddsmoothclip(x,e);
endfunction

analog function real smoothsign;
    real x,e; input x,e;
    smoothsign=2*(smoothstep(x,e)-1);
endfunction

analog function real dsmoothsign;
    real x,e; input x,e;
    dsmoothsign=2*dsmoothstep(x,e);
endfunction

analog function real smoothmin;
    real x,y,e; input x,y,e;
    smoothmin=0.5*(x+y-smoothabs(x-y,e));
endfunction
```

```
analog function real dsmoothminx;
    real x,y,e; input x,y,e;
    dsmoothminx=0.5*(1-dsmoothabs(x-y,e));
endfunction

analog function real dsmoothminy;
    real x,y,e; input x,y,e;
    dsmoothminy=0.5*(1+dsmoothabs(x-y,e));
endfunction


analog function real smoothminy;
    real x,y,e; input x,y,e;
    smoothminy=0.5*(1+smoothabs(x-y,e));
endfunction

analog function real smoothmax;
    real x,y,e; input x,y,e;
    smoothmax=0.5*(x+y+smoothabs(x-y,e));
endfunction

analog function real dsmoothmaxx;
    real x,y,e; input x,y,e;
    dsmoothmaxx=0.5*(1+dsmoothabs(x-y,e));
endfunction

analog function real dsmoothmaxy;
    real x,y,e; input x,y,e;
    dsmoothmaxy=0.5*(1-dsmoothabs(x-y,e));
endfunction

analog function real smoothswitch;
    real a,b,x,e; input a,b,x,e,cof;
    begin
        cof=smoothstep(x,e);
        smoothswitch=a*(1-cof) +b*cof;
    end
endfunction

analog function real dsmoothswitcha;
    real a,b,x,e; input a,b,x,e,cof;
    begin
        cof=smoothstep(x,e);
        dsmoothswitcha=1-cof;
    end
endfunction

analog function real dsmoothswitchb;
    real a,b,x,e; input a,b,x,e,cof;
    begin
        cof=smoothstep(x,e);
        dsmoothswitchb=1-cof;
    end
endfunction


analog function real dsmoothswitchx;
    real a,b,x,e; input a,b,x,e,cof;
    begin
        cof=dsmoothstep(x,e);
        dsmoothswitchx=(-a+b)*cof;
    end
endfunction
```

```
analog function real safeexp;
     real x,x1,maxslope,breakpoint; input x,maxslope;
     begin
          breakpoint=log(maxslope);
          x1=x-breakpoint;
    safeexp=exp(x*(x<=breakpoint))*(x<=breakpoint)+(x>breakpoint)*(maxslope+maxslope*x1);
     end
endfunction

analog function real dsafeexp;
     real x,x1,maxslope,breakpoint; input x,maxslope;
     begin
          breakpoint=log(maxslope);
          x1=x-breakpoint;
   dsafeexp=exp(x*(x<=breakpoint))*(x<=breakpoint)+(x>breakpoint)*(maxslope+maxslope*x1);
     end
endfunction

analog function real safesinh;
     real x,maxslope; input x,maxslope;
     safesinh=0.5*(safeexp(x,maxslope)-safeexp(-x,maxslope));
endfunction

analog function real dsafesinh;
     real x,maxslope; input x,maxslope;
     dsafesinh=0.5*(dsafeexp(x,maxslope)-dsafeexp(-x,maxslope));
endfunction
```